NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SIMULATION OF A PARALLEL PROCESSOR
BASED SMALL TACTICAL SYSTEM

by

Panurit Yuktadatta

December 1991

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited.

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School 6b Office Symbol *(If Applicable)*	7a Name of Monitoring Organization Naval Postgraduate School
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization 8b Office Symbol *(If Applicable)*	9 Procurement Instrument Identification Number
8c Address *(city, state, and ZIP code)*	10 Source of Funding Numbers Program Element Number | Project No | Task No | Work Unit Accession No

11 Title *(Include Security Classification)*
Simulation of A Parallel Processor Implementation of Small Tactical System

12 Personal Author(s)
Yuktadatta, Panurit

13a Type of Report	13b Time Covered	14 Date of Report *(year, month,day)*	15 Page Count
Master's Thesis	From January 91 To December 91	December 1991	151

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or positio of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Transputer, Small Tactical System

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Simulation of a Parallel Processor Based Small Tactical System is a part of The Parallel Command and Decision Syste (PARCDS) laboratory, which was established to support research for the Navy's AEGIS combat system. The current U.S. Navy AEGIS combat system uses the standard AN/UYK-7 computer, which has four processors in the computer system. In probably le than one decade they will not be capable of handling the increasing demand for some complex software systems. Military comma and decision systems of the next decade must be characterized by economy, speed, stability, reliability, and ease of repair. T transputer features all of these benefits and provides a scalable network of transputers which is relatively easy to design. The need f parallel processing grows more evident daily since the best high-performance uni-processor architectures are reaching their limits. T prime objective of this thesis is to model a Small Tactical System by using a network of transputers to develop the transputer versi of the Ada programming language implementation of a Small Tactical System.

20 Distribution/Availability of Abstract	21 Abstract Security Classification	
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified	
22a Name of Responsible Individual Uno R. Kodres	22b Telephone *(Include Area code)* (408) 646-2693	22c Office Symbol CS/KR

DD FORM 1473, JUN 86 Previous edition are obsolete security classification of this pa

S/N 0102-LF-014-6603 Unclassifie

Approved for public release; distribution is unlimited.

Simulation of A Parallel Processor Based Small Tactical System

by

Panurit Yuktadatta
Lieutenant, Royal Thai Navy
B.S.(Electrical Engineering), Royal Thai Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1991

ABSTRACT

The Simulation of A Parallel Processor Based Small Tactical System is a part of The Parallel Command and Decision System (PARCDS) Laboratory, which was established in early 1980's to support research for the Navy's **AEGIS** combat system.

Current U.S. Navy's AEGIS system using the standard AN/UYK-7 computers, which has four processors in the computer system. When one of them fails, the system automatically reloads the remaining three processors with software that has a reduced capability. But in probably less than one decade, they will not be capable of handling the increasing demand for some more complex software systems.

Military command and decision systems of the next decade must be characterized by economy, speed, stability, reliability, and ease of repair. The transputer features all of these benefits and provides a scalable network of transputers which is relatively easy to design. The need for parallel processing grows more evident daily, since the best high-performance uniprocessor architectures are reaching their limits.

The prime objective of this thesis is to model a small tactical system by using a network of transputers to develop the transputer version of the Ada programming language system which models a small tactical system.

THESIS DISCLAIMER

This thesis use a number of names which are trademarks of various corporations. In this section we give the appropriate credits.

- Ada is a trademark of the United States Government Ada Joint Program Office.
- Transputer and OCCAM are trademark of the INMOS Limited, United Kingdom.
- Alsys-Ada is a trademark of Alsys Limited, United Kingdom.
- MS-DOS is a trademark of Microsoft Corporation.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defence or the U.S.Government.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

We owe a great deal to many people who have given us help, ideas and encouragement during the process of writing this thesis. Special thanks go to my wife, BOONTHARIKA SARIKHAGANON, for her support and patience during the past two years at the Naval Postgraduate School. Also, we would like to mention a few persons individually at this time:

Dr. Uno R. Kodres (NPS, Monterey)

Dr. Se-Hung Kwak (NPS, Monterey)

Mr. John Locke (NPS, Monterey)

Capt. Patrick Barnes, USAF (NPS, Monterey)

LCDR. Jeff Schweiger, USN (NPS, Monterey)

Lt. John Hartman, USMC

CDR. Gilberto F. Mota, Brazil Navy

I. INTRODUCTION

A. BACKGROUND

1. Historical Background

Since World War II, the development of radar has produced spectacular changes in the conduct of naval warfare. Jet aircraft replaced their propeller driven predecessors. The increasing speed of attackers such as aircraft or missiles was seen as a reason for a requirement of some type of system or technique to gather operational data, process the data into meaningful information, and present the information as fast as possible in order to make the intercept decisions.

In the late 1950's and early 1960's, with the inception of the Navy Tactical Data System (NTDS), the U.S. Navy began using computers in tactical shipboard systems. The Naval Tactical Data System evolved from this need to normalize and convert increasing amount of information from multiple sources to common representations that could be processed, stored, and disseminated to shipboard tactical users so that weapons employment decisions could be made more effective with less reaction time. [Ref.1]

In the late 1970's the developers had the primary mission of investigating alternative architectures for the Aegis combat system. Aegis, named after the shield of Zeus, was originally designed for the U.S. Ticonderoga class (CG-47) guided missile cruiser, whose central unit is the three dimensional Phase Array Radar AN/SPY-1, the four processors computer system AN/UYK-7 and the weapons control system.

In early 1980's the Parallel Command and Decision System (PARCDS) Laboratory was established to support research for the Navy's Aegis combat system. The early research involved tightly connected single-processor systems modelling parallel processing. The following decade brought many advances in computing power, especially in the field of parallel processing. [Ref.2].

2. The Aegis Combat System

The Aegis Combat System is the most powerful and complex combat system available to the surface fleet of the U.S.Navy. The Aegis Combat System, designed for guided missile cruiser, was engineered as an integrated system of computer programs, sensors, and weapons to provide a multi-warfare capability. [Ref.3].

Aegis consists of three major system components. These are the powerful phase array multi-function radar, AN/SPY-1, the command and decision system (C&D), and the weapons control system (WCS). The SPY-1's function is to detect targets, C&D performs command, control and communication functions. The WCS function is to evaluate the engagement, provide and execute fire control solutions as well.

Since the technology grows rapidly, the U.S.Navy keeps trying to improve the flexibility, reliability of the Aegis Combat System by investigating the possibility of replacing the current components with the new fashion ones. The most important variable in the development of Aegis Combat System is the reaction time. For example, there was the investigation of replacing current communication network for the Aegis combat system aboard Naval ships with dual optical fiber rings and the investigation of replacing the old expensive AN/UYK-7 computers with parallel computer networks.

The next few decades will see the design, development and deployment of the next generation surface combatant. It will most likely be built upon the current Aegis computer system architecture, expanding to a well integrated, highly reliable and easily operable real-time combat system.

3. Transputer Review

The term "transputer" is an acronym for "transistor computer" where it reflects the ability of this device to be used as a system's building block, much like the transistor was in the past. The nice feature of the transputer is that it adds a new level of abstraction, which provides a very simple way to design a concurrent system. As a formal definition we could state that the transputer is a single-chip microcomputer that has its own local memory and four communication links. The links may be thought of as small special purpose processors which steal no cycles from the main CPU, in such a way that we could have all four links and the CPU working at the same time, without degrading the performance of the program's execution. [Ref.4].

The transputer is a parallel microprocessor, generally categorized as a Multiple Instruction Multiple Data (MIMD) computer. This means that transputers are used to execute different operations on separate data at the same time. A transputer operates as a stand-alone machine or as a processing element interconnected by its links to from a computing array or network of transputers. [Ref.5].

The INMOS IMS T800 transputer includes a fast processor, a floating-point processor, memory, and communication facilities in one chip. The IMS T800 is designed for communication in a network of transputers. Each transputer has four link engines,

each of which supports a bidirectional communications link that operates as a direct memory access (DMA) channel. Transmission rates of 20 Mbps per link may be achieved concurrently as the T800 processor continues at high speed. Our estimates are that the T800 computes at about ten times faster than the AN/UYK-7. According to the performance measurement of J. Dongarra [Ref.9], the T800 is 2.3 times faster than the INTEL 80386/80387 combination and four times faster than the MOTOROLA 68020.

B. ENVIRONMENT

1. Hardware Environment

The designed network of transputers which model the Small Tactical System using the IBM-compatible (Zenith Z-248) microcomputer that hosts the IMS T800 25 Mhz transputer on the TMB 08 motherboard (interface between host transputer and PC) with four external transputers, as shown in Figure 1.

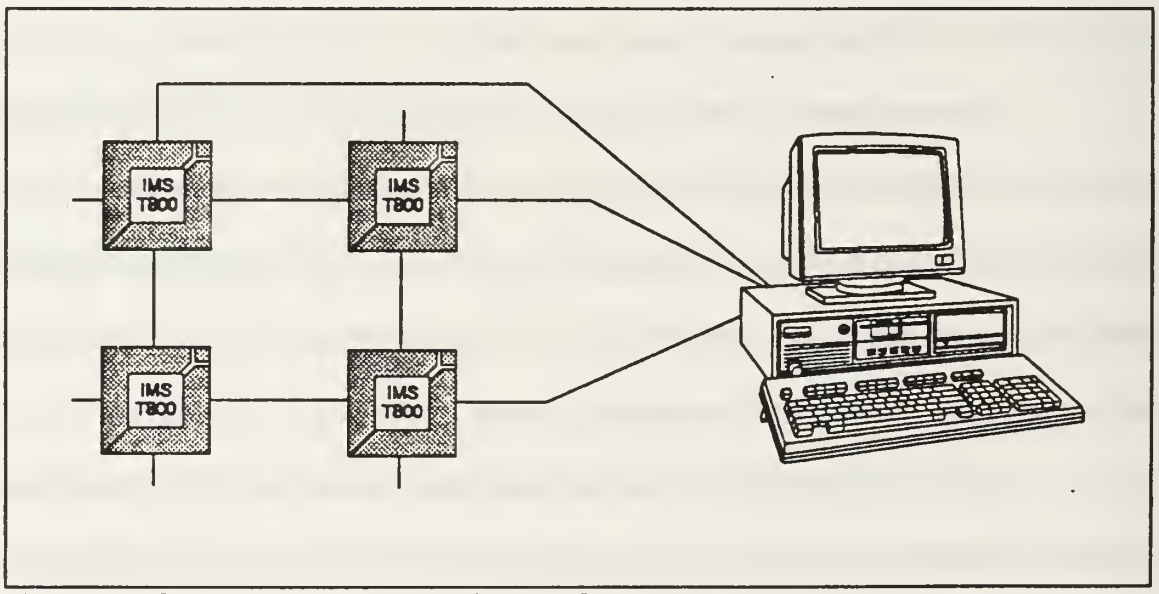

Figure 1: PC Host with Network of the IMS T800 Transputers

2. Software Environment

a. ADA Programming Language

The Ada language is currently undergoing revision, Ada 9x as a new standard Department of Defence (DoD) programming language. Ada will be the standard language for programming computers that are components in weapons systems. The language incorporates the latest concepts in algorithmic language design, including modern control structures, user defined data type, generic units, exception handling, and the ability to coordinate concurrently executing "tasks".

It is asserted that Ada can be used as a specification tool. This results from the ability of Ada to allow top-down program development, in which functionality can be omitted at a high level, only to be incorporated at a later time. In addition, Ada incorporates control structures for multi-tasking, in which concurrently executing tasks are coordinated in a special way. Information may be passed among the tasks in a manner which is independent of the underlying computer architecture. [Ref.7].

b. Alsys-Ada Compilation System

In October 1989, **Alsys** produced the first compiler capable of supporting multi-processor programming in Ada. The PARCDS laboratory purchased this compiler for the transputer systems. Our experience in using the Alsys-Ada compiler's ability to generate Ada code for a single processor found that the compilation process is a complex task, even in the uniprocessor case. In spite of this complexity in this thesis, the Alsys-Ada is selected to be the compilation tool for Ada programs in a multi-transputer network.

c. OCCAM 2 Toolset

The Occam 2 toolset is a set of software tools for developing transputer programs on host systems. Used with the occam libraries, it provides a complete environment for developing programs on transputers and transputer networks. The toolset allows Occam programs to be written using any convenient text editor. Application programs are then compiled and linked using Occam programs resident on the host or running on transputer board. Self-booting code for single transputers and multi-transputer networks is produced and loaded from the host system down the transputer links.

d. MAKE Program Maintenance Utility

MAKE is a utility program designed to assist in the automatic updating and regeneration of computer programs. MAKE provides a specialized script language and an interpreter designed to facilitate control of the programming environment. MAKE program automates the process by determining which parts of the program have been changed since the last compilation and rebuilds them accordingly.

MAKE operates by processing a programmer prepared script file named "makefile". MAKE executes a list of information provided in the makefile. Makefile contains entries that describe how to bring a target object code up to date with respect to those on which it depends, called "dependencies" along with a list of macro definitions for commands needed to rebuild the modules or programs. This information tells MAKE which system commands it should issue to process individual files. Since dependency is a target, it may have dependencies of its own. Targets and dependencies comprise a tree structure that makes traces when deciding whether or not to rebuild a target.

MAKE recursively checks each target against its dependencies, beginning with the first target entry in makefile if no target argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing or to be older than any of its dependencies, MAKE rebuilds it. To build a given target, MAKE executes the list of commands called a rule. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by MAKE.

C. THESIS OBJECTIVE

The basic thrust of this thesis is that the same software running under the old expensive AN/UYK-7 computer could run equally well in the commercially available VLSI microprocessors. And as expected, in probably less than one decade, the old fashioned Navy's standard computers will not be capable of handling the increasing demand for some more complex software systems.

This thesis is a part of the Parallel Command and Decision System Laboratory, whose researchers investigate the possibility of replacing the old standard Navy's computers for the Aegis real-time combat system aboard Naval ships with the network of transputers in order to reduce the reaction time of the Command and Decision Systems.

The objective of this thesis is to try to keep up with the upcoming new technologies especially, the TRANSPUTER. The investigation has been made by modelling the Small Tactical System with the network of transputers, based on the standard DoD programming language, ADA, as the programming language. The Alsys-Ada compilation system is used in the designed network of transputers.

D. THESIS ORGANIZATION

This thesis is presented in five chapters and eight appendixes. Chapter I was the historical background, the development of the Navy's AEGIS combat systems and the introduction to the new fashion of the commercial VLSI microprocessor, TRANSPUTER.

Chapter II provides the general idea of the designed transputer network implementation of the Small Tactical System, and the hardware and software environment used in the designed network.

Chapter III describes the design of each subsystem in the modelling of the Small Tactical System. This chapter focuses on the underlying design concepts used in reaching the objective of each subsystem.

Chapter IV focuses on the software development in Ada by using the Alsys-Ada compilation system and the Occam 2 toolset. Executing the program on the network of transputers to perform the functions of the Small Tactical System, and to demonstrate the communication of data among those transputers through their links.

Chapter V states the conclusions and recommendation for future research.

II. SMALL TACTICAL SYSTEM MODELLING

A. SIMULATION OF SMALL TACTICAL SYSTEM

1. General Idea

After a target has been detected, the decision is made either to attack or not. If an attack decision is made, then the target is tracked. A future position is estimated, and a weapon is launched to intercept the attacker at the predicted interception point. To clarify the approach, a five transputer network is designed to solve the problem.

2. The Designed Network

The designed network of modelling the Small Tactical System using the IMS T800 25 Mhz transputer as the host transputer and 4 IMS T800 20 Mhz external transputer as the subsystems as shown in Figure 2.

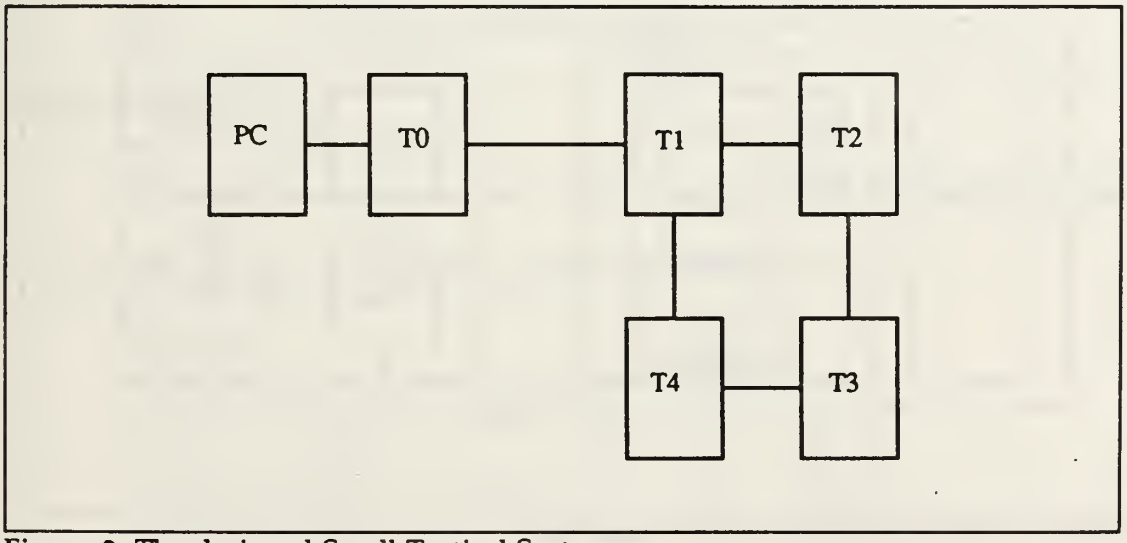

Figure 2: The designed Small Tactical System

T_0 is the host transputer which performs the Human Interaction.

T_1 performs as the Target Tracker Subsystem.

T_2 performs as the Target Prediction Subsystem.

T_3 performs as the Ballistic Interception Subsystem.

T_4 is a hot spare to make the system Fault Tolerant.

3. The Designed network Configuration

The designed multi-transputer network implementation of the Small Tactical System is required to have the transputer boards as:

- the IMS B417 TRAM (TRAnsputer Module)
- the TMB08 TRAM Motherboard
- the IMS B003 Evaluation Board

the network has been connected as shown in the Figure 3.

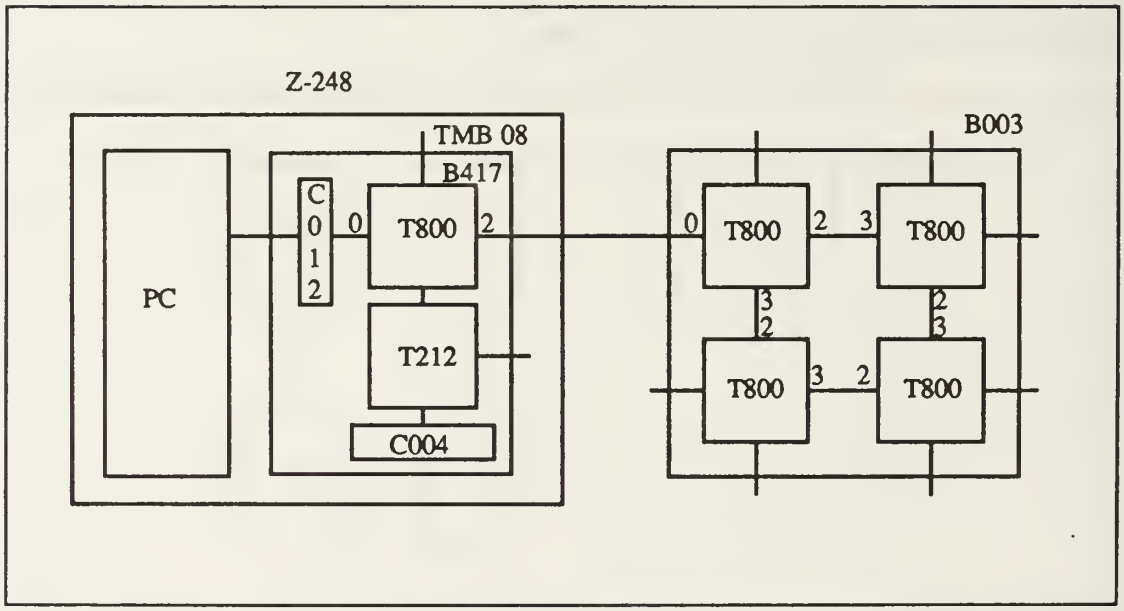

Figure 3: Connection Network of Transputers

The connection of the designed network is made by using two soft wires. One is connected the down subsystem from the TMB08 to upload a program at the B003 board and the other one is connected link 2 of the T800 25 Mhz in the TMB 08 board to link 0 of the first T800 20 Mhz in the B003 evaluation board as shown in Figure 4.

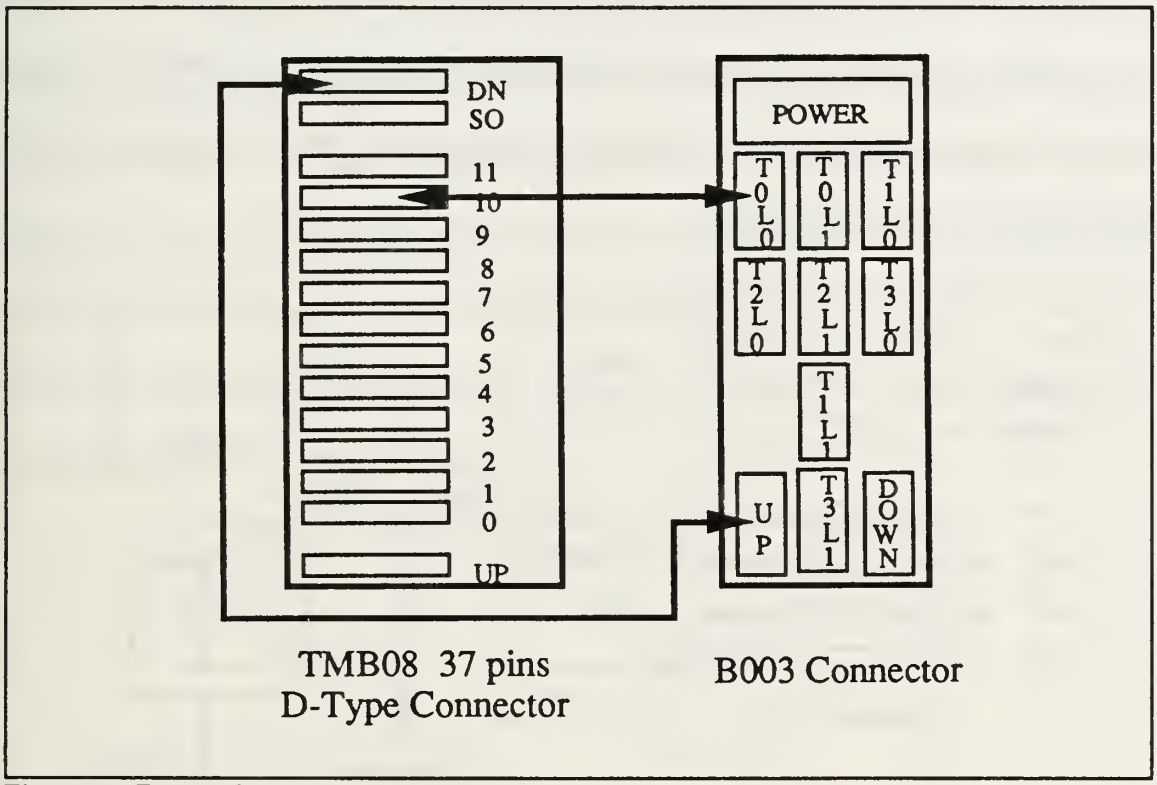

Figure 4: External Links

When all connections have been made, there is a check program that can check the connections. Output of the check program will look like below

```
check 1.21
#  Part   rate  Mb    Bt [ Link0  Link1  Link2  Link3 ]
0  T800d -25  0.18  0  [ HOST   1:1    2:0     ...   ]
1  T2    -17  0.90  1  [  ...   0:1     ...    C004  ]
2  T800c -20  0.90  0  [  0:2    ...     3:3    4:2   ]
3  T800c -20  0.90  3  [  ...    ...     5:3    2:2   ]
4  T800c -20  0.90  2  [  ...    ...     2:3    5:2   ]
5  T800c -20  0.88  3  [  ...    ...     4:3    3:2   ]
```

B. TRANSPUTER BOARDS

1. The IMS B417 TRAM (TRAnsputer Module)

The IMS B417 is a TRAnsputer Module (TRAM) incorporating a IMS T800 25 Mhz transputer, 64K bytes of static RAM and 4K bytes of dynamic RAM. The 4M bytes of DRAM is sufficient to run the Ada compiler from Alsys. The IMS B417 is board level transputer with a simple, standardized interface. The circuit diagram of the IMS B417 TRAM is shown in Figure 5.

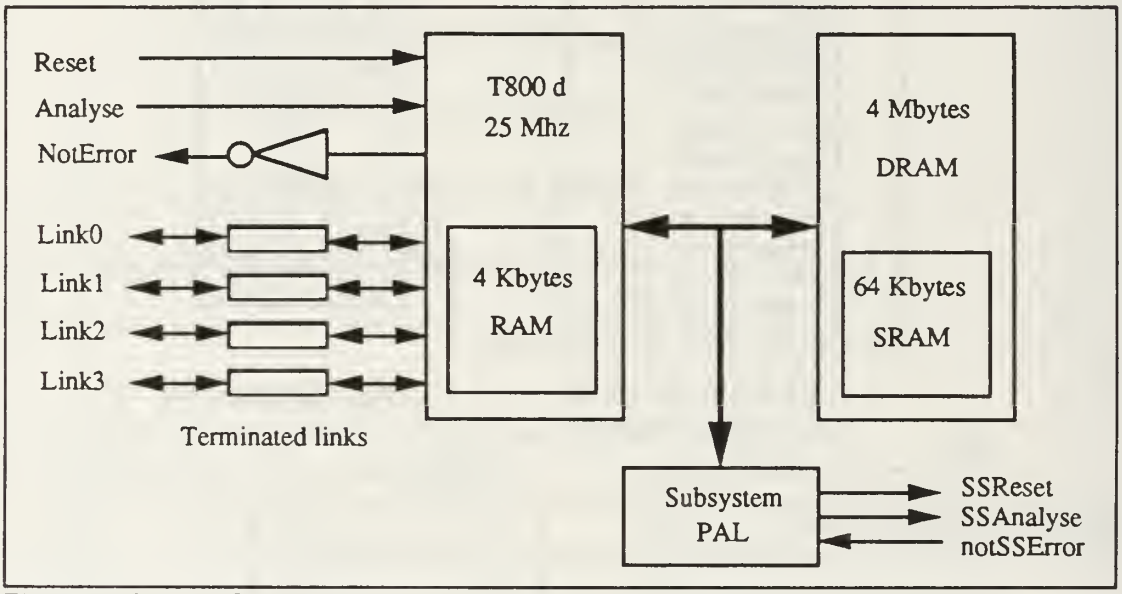

Figure 5: The IMS B417 TRAM

a. The IMS T800 25 Mhz Transputer

The IMS T800 25 Mhz transputer is a 32-bit CMOS microcomputer with a 64 bit floating point unit and graphic support. It has 4K Bytes on-chip RAM for high speed processing, a configurable memory interface and four communication links. It is able to perform floating point operations concurrently with the processor. The 32 bit wide

memory interface uses multiplexed data and address lines and provides a data rate up to 40M bytes/s. A configurable memory controller provides all timing, and DRAM refresh signals for a wide variety of mixed memory systems.

b. Memory Configuration

The IMS B417 is able to access 4M bytes of memory. This is comprised of 4K bytes of internal transputer memory, 60K bytes of external SRAM and 4032K bytes of external DRAM. There are, in fact, 64K bytes of SRAM components and 4M bytes of DRAM components on the module, but the address space of each type of memory are superimposed. Therefore, the total memory available is limited to 4M bytes which is sufficient to run Ada compiler product of Alsys. The start addresses of the different types of external memory on the IMS B417 is shown in table below :

	Hardware bytes address
SRAM From :	# 80010000
To :	# 8000FFFF
DRAM From :	# 80010000
To :	# 803FFFFF

2. The TransTech TMB08 TRAM Motherboard

The TMB08 is a TRAM motherboard that enables users to build multi-transputer system that can be plugged into an IBM PC-XT or PC-AT. It has slots for up

to ten TRAMs. By using an IMS C004 programmable link switch, incorporated on the TMB08 allows a large variety of networks to be created under software control. The IMS C004 link switch is controlled by the T212 16-bit transputer. An interface to the IBM bus is provided so that a program running on the IBM PC can control the TRAMs on the TMB08 board. Different events on the TMB08 can generate an interrupt on the IBM PC and the PC can carry on with other tasks while programs are running on the TMB08. The diagram of the TMB08 is shown Figure 6.

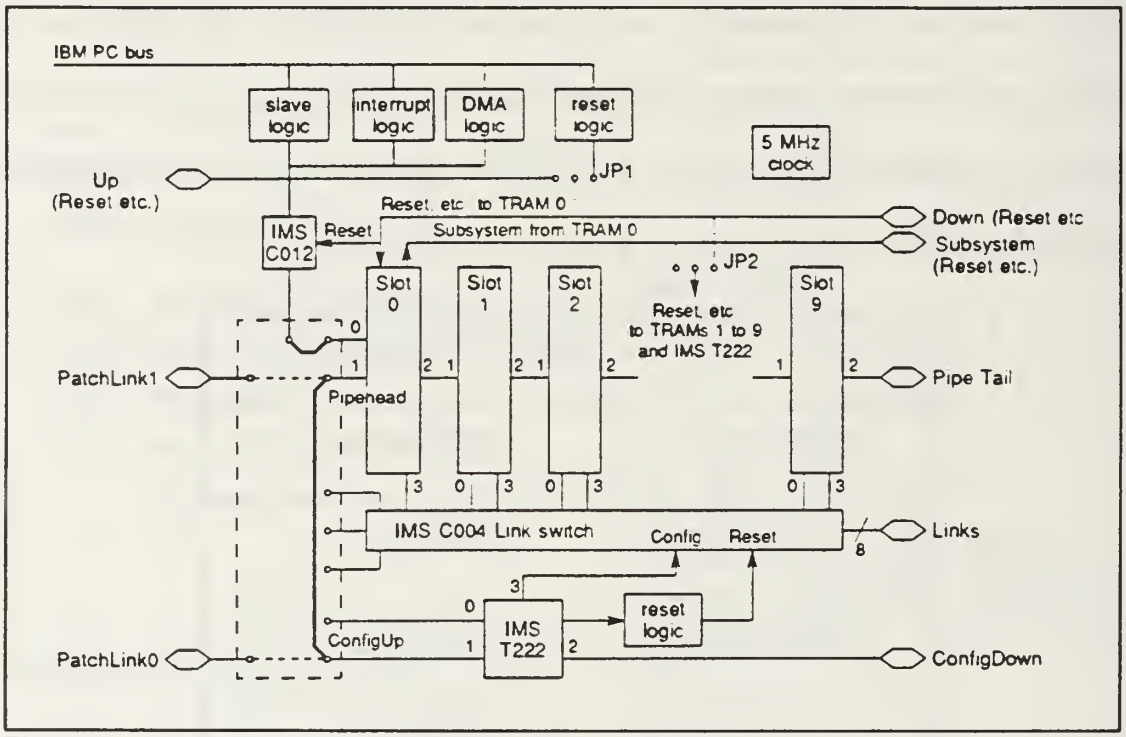

Figure 6: The Transtech TMB08 TRAM Motherboard

a. The IMS T212 Transputer

The IMS T212 integrates a 16-bit microprocessor, four standard transputer communication links, 2K bytes on-chip RAM, a memory interface and peripheral interfacing on a single chip. The processor shares its time between any number of

concurrent processes. A process waiting for communication or a timer does not consume any processor time. The T212 use a DMA block transfer mechanism to transfer message between memory and another transputer product via links. The link interface and the processor all operate concurrently, allowing processing to continue while data is being transferred on all of the links. The 2K bytes of static RAM provide a maximum data rate of 40M bytes/s with access for both the processor and link. The T212 can directly access a linear address spaces up to 64K Bytes, and has a 16-bit wide data bus and a 16-bit wide address bus, non-multiplexed, providing a data rate of up to 20M bytes/s, and supporting word or byte organization.

b. The IMS C004 Programmable Link Switch

The IMS C004 device is a transparent programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs. Any of 32 links may be connected to any other by sending the appropriate control data to the IMS C004 along its configuration link. The configuration link of the IMS C004 is connected to an IMS T212 transputer. Configuration data for the IMS C004 is fed into link 1 of the IMS T212 (ConfigUp), which then passed it on to the IMS C004 on link 3. The same data is also passed out of the IMS T212 through link 2 to the edge of the TMB08 board.

c. The IMS C012 Link Adapter

The IMS C012 link adapter is a universal high speed system interconnect, providing full duplex transputer link communication by converting bi-directional serial link data into parallel data stream. The IMS C012 provides an interface between an

INMOS serial link and a microprocessor system bus. Two interrupt outputs are provided, one to indicate that input data is available and one for output buffer empty. There are two link adapters in the TMB08 board, one connects between PC and the IMS B417 TRAM and the other one connects between PC and the IMS T212 transputer. In the TMB08 motherboard, the status of the registers of IMS C012 are continuously polled by software running on PC.

3. The IMS B003 Evaluation Board

The IMS B003 evaluation board (T800 20 Mhz version) is a double extended Eurocard containing four T800 20 Mhz transputers. The four transputers are synchronized by using the same clock in the logic shared, as shown in the Figure 7.

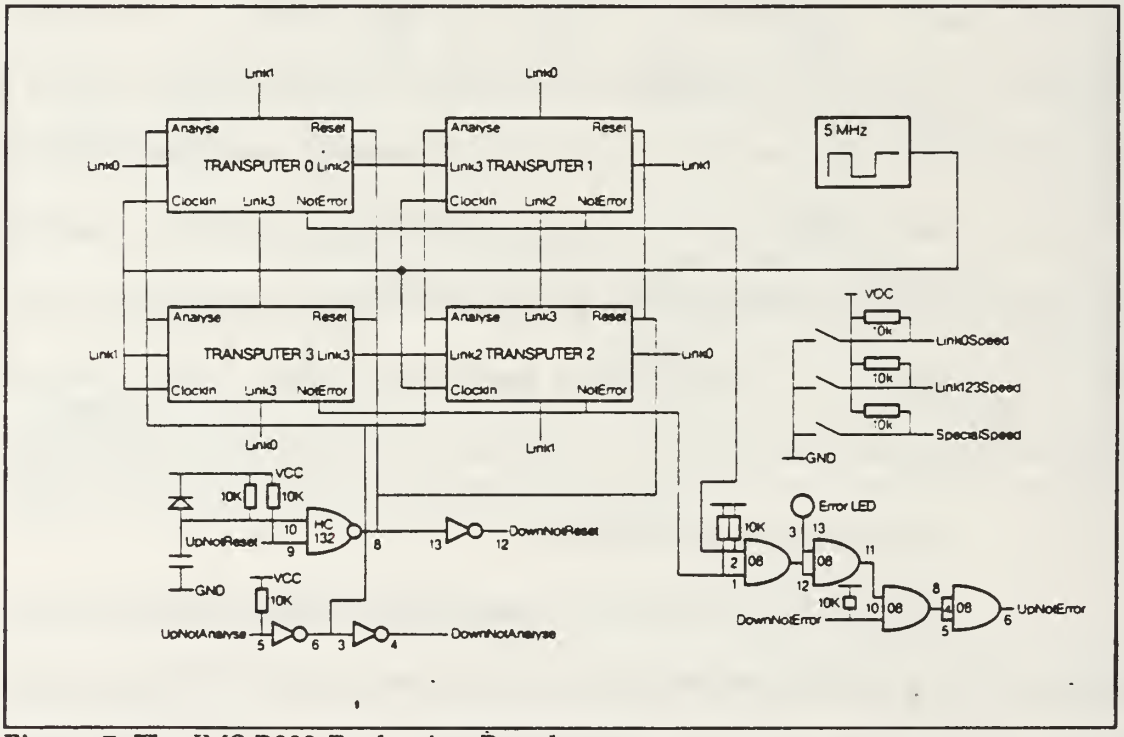

Figure 7: The IMS B003 Evaluation Board

a. The IMS T800 20 Mhz transputer

The IMS T800 20 Mhz transputer is a 32-bit microprocessor with a 64 bit floating point unit, four standard transputer communications links, 4K bytes of on-chip RAM and a memory interface on a single chip. The T800 20 Mhz provides high performance arithmetic units and microcode support for floating point operations. The processor shares its time between any number of concurrent processes. The link interfaces and the processor all process concurrently, allowing processing to continue while data is being transferred through all of the links. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of 26.6M bytes/s. The IMS T800 20 Mhz is pin compatible with the IMS T414 20 Mhz, if the additional inputs of the IMS T800 are held to ground. The IMS T800 20 Mhz can thus be plugged directly into a circuit designed for a 20 Mhz version of the IMS T414. Software should be re-compiled, although no changes to the source code are necessary.

b. Links

The transputers on the board are connected in a square, the square has rotational symmetry, with link 2 of each transputer connected to link 3 of the next transputer. Link 0 and link 1 of each transputer are taken to the edge connector. The speed of communication between links for internals on board running at 20M bits/s, while Link 0s are connected off the board and running at 10M bits/s. Alternatively, all the links may be set to run at 10M bits/s or 20M bits/s. But if running external links at 20M bits/s, the link connections have to be short. The link connection to the B003 board depend on the designed program.

C. THE ALSYS ADA COMPILATION SYSTEM

The Alsys Ada Compilation System consists of the Compiler and Binder, operating in the Alsys Multi-Library Environment. The Compiler generates executable code for T4 or T8 transputer targets. Multi-Library Environment provides a powerful way of managing Ada development efforts. It allows compilation units to be flexibly shared among libraries, and eliminates the need to copy library units to share them, along with the associated version control problems.

1. The Compiler

The Compiler requires two names as input: the source file, which contains the Ada source code and a program library in which to place the compiled object units. Compilation units in the source file must be specified in a valid order such that if a unit depends upon other units, then those other units must come before that unit. If the order of the units is invalid, or if a unit named in a **with** clause does not exist, the Compiler will issue an error message and the compilation will fail. The output from the Compiler normally consists of an update version of the designated program library, containing the compiled object code for the units included in the source file and a compilation listing detailing the results of the compilation.

The compilation listing always includes diagnostic message for any errors detected during compilation. There are options to include in the listing both the source code of all compiled units and the object code. The listing may be sent to a standard output device or a file to which the listing is to be written. Alternatively, the compiler can perform an error analysis without generating any object code. If this is the case, only a

a compilation listing will be produced and the program library will not be modified in any way. The error analysis can be restricted to syntactic errors only, or both syntactic and semantic error analysis can be requested.

2. The Binder

The Ada Binder combines the required units of an application program into an object module. The input to the Binder consists of an Ada program library containing the main unit of the Ada program and the name of the main program unit within this library. The Binder uses the main program name supplied together with the dependence information in the program library to find the compilation units needed by the program. Other related units can be contained in other libraries to which links are established from the current library (these related libraries may then refer to future libraries and so on).

The Binder produces two files as output: an object module containing the code and data for all the Ada compilation units included in the program. Default name is *"program.O"* and a listing of file summarizing the results of the binding process. The default name of this file is *"program.BND"*. The listing file includes any error message output by the Binder. The listing may optionally include information and warning message, information about the composition of the program, the elaboration order of the program's compilation units and the names of any subprograms which are uncalled in the program and have thus been deleted. If names are not specified for the Binder output file, default file names based on the main program name will be used. If errors are detected during binding, no object module will be produced.

3. Linking, Loading and Executing

The tasks of linking, loading and execution are performed using programs provided as part of the *Occam 2 toolset,* a set of tools supplied by INMOS to aid with the development of transputer based applications.

a. The Occam 2 Toolset

There are a number of different implementations of the toolset running on a variety of host computers. However, the basic facilities provided by the toolset are the same regardless the implementation. The tools which are of particular interest are:

ilink. collects together all the code for a program resolving.

iboot. A tool which adds a bootstrap code to link programs prior to loading the program on a single transputer.

iconf. A tool which configures a program prior to execution on multi-transputers network.

iserver. The host server, loads bootable program onto a processor.

This implementation of the toolset uses particular command line syntax and file naming. The file naming conventions are particularly evident in the area of file name extensions. Most extensions are composed of single letter code followed by two additional characters, where the single letter identifies the nature of the file:

.cxx Linked code files, the output of *ilink.*

.mxx Module map produced by *ilink.*

.bxx Bootable code file, the output of *iboot.*

.dxx Bootstrap code description produced by *iboot.*

The additional characters *(xx)* serve to identify the transputer type and error mode of compilation. For the Alsys Ada, the error mode used is STOP process (s) mode, so the second additional character is always an "s". The processor type can be any one of "4", "5", or "8" corresponding to a T414, a T425, or any T8 transputer target.

b. Program Linking

For a system of multiple Ada programs, or a system of programs running on multiple transputers, it is likely that you will have to invoke the linker directly rather than have the Binder do this. Linking is accomplished using the *ilink* tool. The linker is given the names of any separately compiled object or library files and produces a single object file which can then be loaded onto a transputer system. *ilink* accepts commands with the following syntax

ilink { inputfile } { option }

For the current implementation the complete list of input files which must be supplied to the linker in addition to the Binder output file is as follows:

- harness.t8s. The *occam harness* used to invoke the Ada program.
- adarts8.lib. A library of routines which form the part of the Run Time Executive which has not been implemented in Ada.
- occam8s.lib. The *occam compiler library* which is provided as the Occam 2 toolset for some of the occam routines in adarts8.lib to reference.
- hostio.lib. Another library from the Occam 2 toolset; it provides access to the facilities of the *iserver*.

c. Program Loading and Execution

Although the linker produces a single object file, there is still a step required before the program can be executed. It is necessary to distributed the program among the network of transputer. For multi-transputer programs, executable code is produced using the configuration tool *iconf*.

D. THE ENVIRONMENT OF AN ADA PROGRAM

1. Interface to Host System

All access to the services of the host MS-DOS operating system is via an interface conforming to the INMOS file server. The server is used to invoke a program to be run on the transputer and continues execution on the host (MS-DOS) while the program is running.

Communication between the server and the program is via a pair of channels; one channel is used for requests by the program and the other for responses from the server. For single processor programs these channels are passed as parameters to the program. For multi-processor programs, the channels are accessed via the physical links on the root transputer. Although the server is a single resource which can only be used by a single process running on the root transputer, it is possible to share the server resource using a multiplexor process. The multiplexor takes the channels request from many processes wishing to use the server and maps them onto the actual server channels, ensuring that the communications between the server and its clients do not overlap.

2. Interface to Other Languages

a. Ada Program Interface

An Ada program may be called in much the same way as a normal occam procedure. All Ada programs conform to the following occam specification:

```
PROC Alsysadaentrypoint ( [] INT ws1, in out, ws2)
```

The name "Alsysadaentrypoint" is the default entry point name of an Ada program. The default occam harness contains a #IMPORT directive referring to the file alsys.tax which consists of a dummy occam procedure conforming to the above specification. This technique allows the default occam harness to be used with any Ada program, avoiding the need to change the #IMPORT directive and thus re-compile the harness for each program.

It is possible to specify an alternative entry point name using the Binder option ENTRY_POINT. When this option is used together with the Binder option LEVEL=BIND, a customized occam harness can include a #IMPORT directive referring to the output file of the Binder (the ".O" file) and invoke the Ada program using the entry point name specified.

The in and out parameters are vector of pointers to channels going to and coming from the Ada program. If there is a single workspace (that is, there is no stack memory), the ws1 parameter is used as the main workspace and the ws2 parameter is unused. If there are two workspaces, the ws1 parameter is used as stack memory and the ws2 parameter is used as the main workspace. Stack memory is an area which is assumed to map onto the low-addressed fast internal memory of the transputer. If insufficient

workspace is passed by the occam harness, the Ada program will be terminated immediately and a message issued along the error channel. Four of the channel parameters of an Ada program are reserved for use by the run-time system:

- out[0] is used as an error channel.
- in[0] is reserved although currently unused.
- out[1] is used for the requests from the Ada program to the server.
- in[1] is used for the communication of responses from the server.

All Ada input-output operations are accomplished by issuing requests to the server and therefore make use of the channels reserved for such communication. The following routines from the host file server library, *hostio.lib*, are potentially called from a program using the predefined input-output package of Ada:

so.open, so.close, so.read, so.write, so.gets, so.puts, so.remove, so.time

Note that the standard input and standard output files of TEXT_IO are mapped onto the standard input and standard output streams of the server. The error output from the Ada program is treated as a special case and is directed to the error channel rather than making use of the server directly. Severe error situations are reported along this channel, including program deadlock and the unhandled exceptions.

b. Occam Calling Ada - the Occam Harness

In order to integrate Ada with other languages a well defined interface is required. Although the Ada program interface is adequate, a simpler interface is possible if the program could be treated as a true occam process. Ada programs may then be run in parallel on a single processor or distributed across a multi-transputer network,

just as occam processes. To achieve this simplicity of interface, an occam process called a *harness* is used as a wrapping for the Ada program. A default occam harness is provided as part of the Compilation System in both source and compiled forms. The main body of the harness consists of three processes operating in parallel:

- A multiplexor which combines the error output and the output of the Ada program. This process is provided as part of the server library, *hostio.lib*.
- An error channel collector which collects any output from the error stream and routes it to the standard output stream of the server via the multiplexor.
- A process which sets up the input and output channel vectors of the Ada program and then invokes it, informing the other processes upon compilation.

When all three processes have terminated, the server itself is terminated and control is returned to the host. It should be note that the default harness is suitable only for single Ada programs running on single transputers.

3. Communication Using Transputer Channel

An Ada program can communicate with any other program using transputer channels via the implementation defined package CHANNELS. This program could be running on the same transputer, or on one of its neighboring processors. The CHANNELS package provides access to the channel parameters of the Ada program and to the physical links of the processor on which the program is running. A generic package within CHANNELS provides facilities for communication between programs by using READ and WRITE operation on channels.

a. Using Internal Channels

Channels mapped to transputer links are known as *hard channels*. Processes communicating with each other on the same transputer use internal channels, known as *soft channels,* implemented in the transputer's memory. These internal channels are represented in Ada by objects of the type CHANNEL_TYPE declared in package CHANNELS. Any number of such channels may be declared in an Ada program and used for communication between tasks.

b. Accessing Physical Links

Access to physical links and the event pin of the transputer on which the Ada program is running via the channel contents declared in package CHANNELS.

c. Communicating Data Across Channels

Channel communication can be achieved using the READ and WRITE procedures of the generic package CHANNEL_IO, instantiated with an appropriate type. The following points should be noted when using CHANNEL_IO for communication:

- Object of a task or private type or records containing components of such types should not be passed into or from an Ada program.
- Representation and length clauses should be used to control the layout and size of record objects when communicating with a non-Ada program.
- When a channel is used for communication between two Ada programs, common packages should be used to ensure that each of the programs has a consistent view of the data passed between them.

III. SUBSYSTEM DESIGN AND DEVELOPMENT

A. SINGLE TRANSPUTER SYSTEMS

In the first step of Small Tactical System development, it is necessary to ensure that each subsystem is done correctly. The single transputer system is used to test each subsystem separately.

For single transputer systems, the major issue in the design is that of sharing resources. The resources of a single transputer system are the processor, memory and the server. The occam programming model consists of a number of processes executing in parallel and communicating by the use of channels. The processes which constitute a system may be executing on the same transputer. As a consequence of the model, it is possible for several programs to share the resources of a single processor. Since an Ada program does not interfere with the shared components of the system, one or more Ada programs could form the system. However, the performance (cost) of an Ada program should always be taken into account in the design of a single processor system.

A single transputer system is invoked as a single program (a harness) which is passed to two areas of free memory: a work space and an area of stack memory. It is the responsibility of the programmer to divide these workspaces as appropriate. The workspace allocation is provided in the harness "main.occ" in APPENDIX F.

To implement an application on a single transputer involves four logical steps: Source Compilation, Object linking, Configuring, Loading and Running.

1. Source Compilation

All application source must be compiled for the target transputer. The Alsys-Ada compilation systems permit separate compilation of source units. Once all source units have been compiled, the application can be linked.

a. Source of Occam Harness

The default harness is inadequate only in case where additional channel parameters need to be passed to the Ada program. Each Ada program has its own "mini-harness" which provides a clean interface to the program in terms of the channels used. The main harness is used to invoke each of the mini-harnesses in parallel. Source of the occam harness must have the extension ".occ".

b. Source of Ada Program

All the code written in Ada can be run on a transputer using the Alsys-Ada Compilation system. It can support any standard package written as an Ada program. The application Ada programs must have the filename extension ".ada".

2. Object Linking

For a single Ada program running on a transputer, invoking the Binder alone should be sufficient to produce an object file named "proj.o". Following source compilation, the object binaries are linked together with the relevant run-time library and a proprietary occam support harness. Linking is accomplished using the *ilink* tool. The linker is given the name of any separate compiled object or library files and produce a single object file which can then be loaded onto a transputer system. The support harness

ensures that the application has correct access to the server running on the host platform. The problem which may occur when linking a program for execution on a single transputer is that of name conflict. For Ada programs, this problem can be avoided by changing the entry point name of the program using the Binder option ENTRY_POINT.

3. Configuring

Although the linker produces a single object file, there is still a further step required before the program can be executed. It is necessary to configure the program to prepare it for execution on the target transputer system. Configuration of a single transputer system involves adding bootstrap code to the program using the *iboot* tool. The bootstrap code initializes the processor, allocates workspaces, loads the program code and finally invokes the program. A program invoked by the bootstrap loader should conform to the following specification.

```
PROC main.program (CHAN OF ANY from.server, to.server,
                   []INT workspace, stack.memory)
```

The from.server and to.server parameters are the channels used by the host file server to boot the program and may be used for communication with the server by the program when it is running. The workspace and stack.memory parameters are areas of free memory for use by the application program.

4. Loading and Running

The file produced by *iboot*, which contains the final executable program and bootstrap code, can be loaded onto the target transputer using the host file server, *iserver*. The sequence of operation on a single transputer is shown in Figure 8.

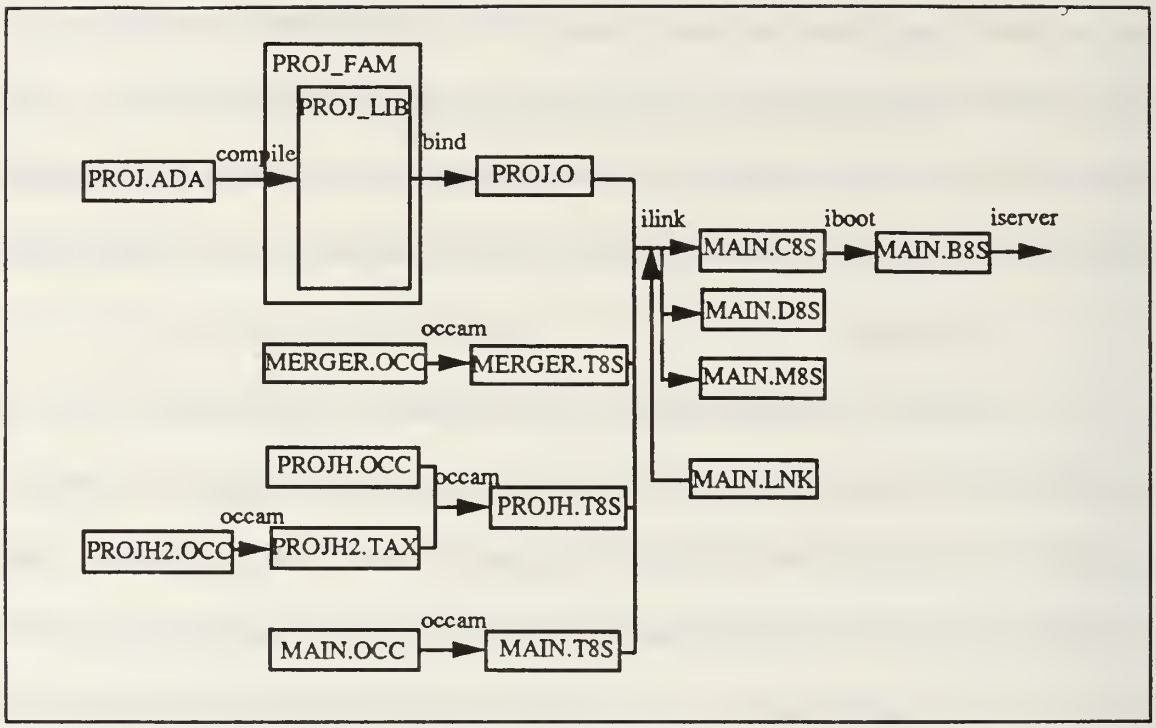

Figure 8: Sequence of operation on single transputer

The following commands, provided in Makefile, are used to build and run the system of Ada programs. It is assumed that the Ada programs have already been compiled and bound such that the object code is available in the file "proj.o".

```
-- Compile separate occam processes.
occam /s /t8 merger.occ
occam /s /t8 projh.occ
occam /s /t8 projh2.occ
occam /s /t8 main.occ

-- Fully link a single step.
ilink /f main.lnk

--Add bootstrap code.
iboot main.c8s

-- Load and run the program.
iserver /sb main.b8s
```

B. HOST TRANSPUTER

1. General Idea

In the combat system, during target engagement by a high performance aircraft or missile, it is critical that the man/machine interface by kept very simple. The display should provide sufficient information with low complexity. Thus, the designed host transputer should perform the communication between the network of transputers and the PC. The system communicates with the operator by keyboard and the monitor screen.

2. The Transputer / PC Host development relationship

The transputer is normally employed as an addition to an existing computer, referred to as the host. Through the host, the transputer application can receive the services of a file store, a screen, and a keyboard as shown in Figure 9.

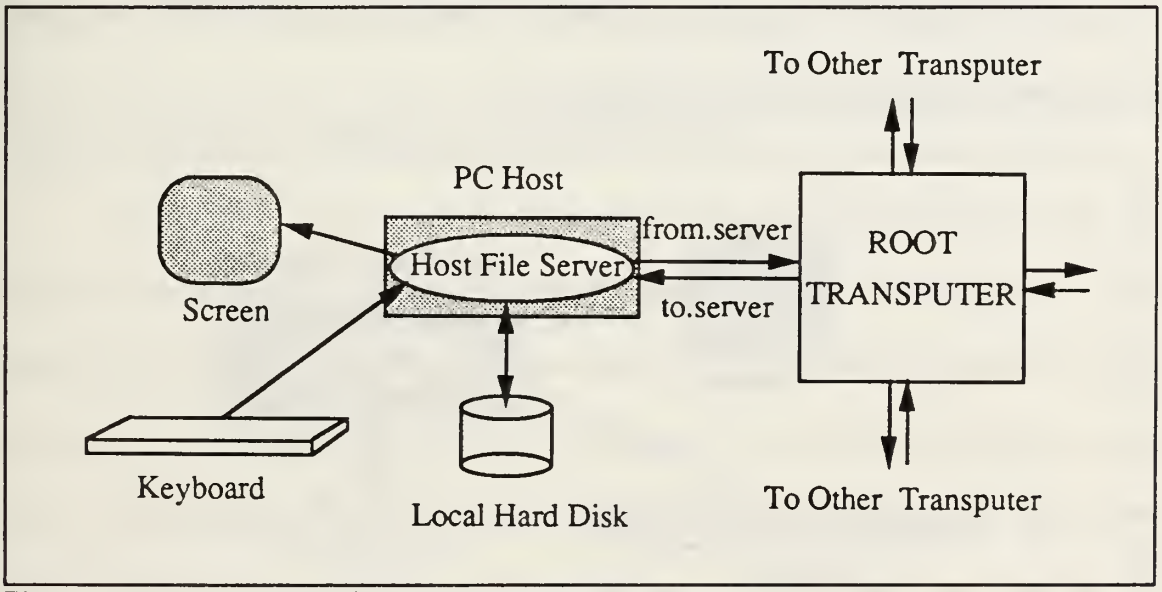

Figure 9: The transputer / host development relationship

The transputer communicates with the PC host along a single INMOS link. A program, called a server, executes on the host at the same time as the program on the transputer network is run. All communications between the application running on the transputer and the PC host services (screen, keyboard and filing resources) take the form of messages. Software written with the Occam toolset and Alsys-Ada compiler, to use the standard INMOS servers, assumes master status in a master/slave relationship between the transputer and PC host. In this situation, messages are always initiated by the transputer system. The root transputer in a network is the transputer connecting to the host bus via the link adapter as described in Chapter II.

3. The Development

The main objective of the Host transputer is to communicate with the PC Host. This section considers the simplest porting situation for an application. Before the porting to transputer the application look like Figure 10.

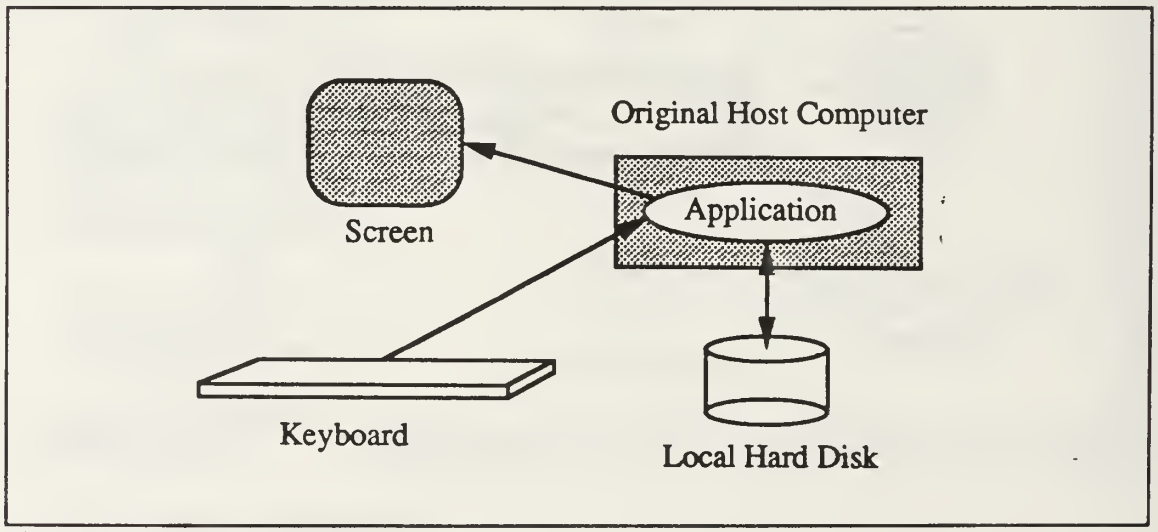

Figure 10: The starting point

No assumptions are made about the nature and capabilities of the original compute engine, except that the application uses only keyboard, screen and file system through standard function calls to the language's run-time library.

When porting to transputer the application is to be lifted from an arbitrary computer system, and executed on a single transputer connected to a supported host platform as shown in Figure 11.

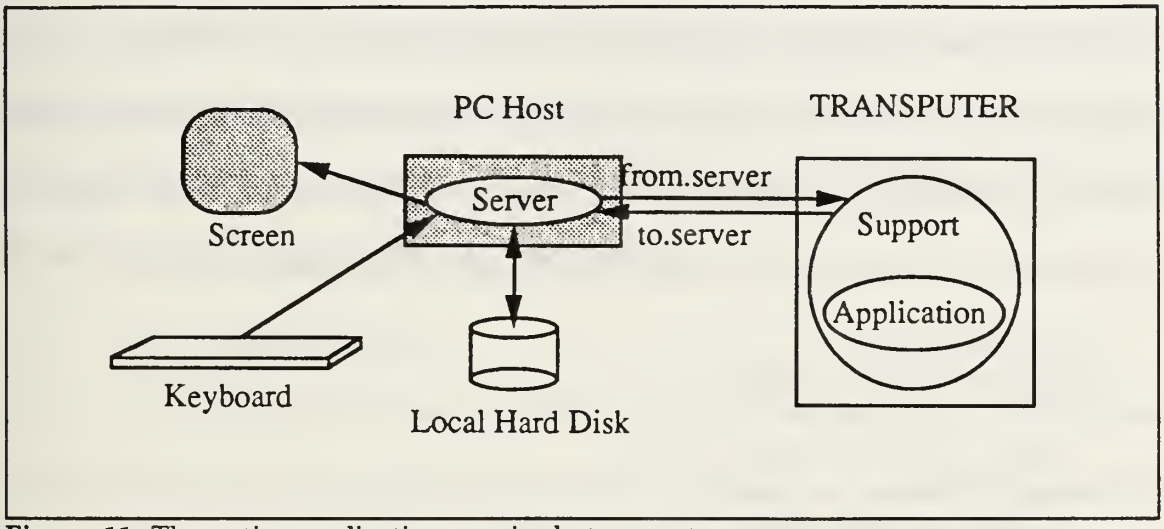

Figure 11: The entire application on single transputer

The PC Host runs a simple program called "server" which ensures that the access requirements of the application in terms of keyboard, screen, and filing, are fully satisfied. The Occam 2 toolsets use a server called *iserver*. The from.server and to.server parameters are the channels used by the host file server to boot the program and may be used for communication with the server by the program when it is running. This server is not recommended for use with application port. The program that controls this operation is in the file main.occ in APPENDIX F.

C. TARGET TRACKER SUBSYSTEM

1. General Idea

In the Navy's Aegis combat systems, a complex real-time radar surveillance system that keeps track of target position, should have a software that supports this operation, package PLANE_TRACKER. It should be able to handle up to 512 planes at a time and provide subprogram to query and update plane positions and velocities. Every time a new plane is detected by radar, PLANE_TRACKER is instructed to start tracking the plane. It is also informed when to discontinue the tracking and should raise exceptions when it cannot handle anymore planes or when an untracked plane is referenced. The specification of a package that is able to handle all the requirement above is:

```
with CALENDAR;
package PLANE_TRACKER is

   MAX_PLANE : constant := 512;
   type MILES is new FLOAT;
   type MILES_PER_HOUR is new FLOAT;
   type PLANE_INFO is record
        X,Y,Z : MILES;
        VX,VY,VZ : MILES_PER_HOUR;
        T : CALENDAR.TIME;
   end record;
   type PLANE_ID is limited private;

   procedure CREATE_PLANE (I:PLANE_INFO; ID:out PLANE_ID);
   procedure REMOVE_PLANE (ID:in out PLANE_ID);
   procedure UPDATE_PLANE (ID:PLANE_ID; I:PLANE_INFO);
   function READ_PLANE (ID:PLANE_ID) return PLANE_INFO;

   ILLEGAL_PLANE,TOO_MANY_PLANES : exception;
private
   type PLANE_ID is new INTEGER;

end PLANE_TRACKER;
```

2. The Tracked Data Simulation

In principle radar system, the procedure of track radar is that the transmitter sends the electromagnetic energy to the target and the receiver receives the echo from that target. The echo information contains the positions and velocities of the target. The processing of the echo information gives range and direction of the target. Due to this principle the tracked data simulation should simulate the echo information.

In order to simulate the tracked data which is required for the Prediction Subsystem, the target trajectory approximation that keeps updating the new position of the target is selected. First, by mathematics, the comparison between Trapezoidal rule and Simpson's rule of integration has been investigated.

a. Trapezoidal Rule

For any trajectory along function y = f(x) from point a to b, the positions in the interval [a,b] can be approximated by the sum of the trapezoids. By dividing the interval [a,b] into subintervals of equal length $\Delta x = (b-a)/n$ and denote the end point of the subinterval by $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ then

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\{f(x_0)+2f(x_1)+2f(x_2)+\ldots+2f(x_{n-1})+f(x_n)\}$$

and the error for the Trapezoidal Rule is

$$|e| \leq \frac{(b-a)^3 M}{12n^2}$$

where M is the maximum value of $|f''(x)|$ on the interval [a,b]

b. Simpson's Rule

The Simpson's rule uses parabolic arcs rather than line segments. By dividing the interval [a,b] into n subinterval (n must be an even integer) with $\Delta x = (b-a)/n$ and denote the endpoints $x_0 = a$, $x_1 = a + \Delta x$, $x_2 = a + 2\Delta x$, ... , $= a + n\Delta x = b$ then

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}\left[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\ldots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)\right]$$

and the error for the Simpson's Rule is

$$|e| \le \frac{(b-a)^5 M}{180n^4}$$

where M is the maximum value of $|f^{IV}(x)|$, the fourth derivative, on the interval [a,b]

c. The method of simulation

The Ada code in APPENDIX A computes the aircraft locations by using Trapezoidal rule and Simpson's rule of integration. It compares the results with the observed long-range radar location and computes error values for each integration method. This program assumes that the aircraft flies back and forth from the position X = 5000 m, Y = 0 m to the position X = 0 m, Y = 5000 m at the altitude of 4000 m. Time of flight is 30 second. The function TRAPEZOIDAL and SIMPSON in this program perform the numeric calculation using Trapezoidal rule and Simpson's rule respectively. Output of this program shows that the position errors of using Simpson's rule are always almost zero when compare with the errors of using Trapezoidal rule. This leads to desire to use Simpson's rule to simulate the tracked data rather than Trapezoid rule because it produces more accurate simulated data.

3. Tracked Data

The Ada code in APPENDIX B produces the simulated tracked positions of the target in three dimensions every one second. In real world, the target also has the velocities in three dimension and those velocities can be changed all the time because of the gravity, wind speed, etc. In this program assumed that the target approaches with acceleration until it reaches its maximum speed, and then all the velocities remain constant. The output shows the positions and velocities of the target every second.

Since the Prediction Subsystem required the three dimensional data at least seven previous position values, the simulation of tracked data should produce the three dimensional position data and send the seven vectors of the three dimensional position values to the Prediction Subsystem in the same time. And also in order to keep updating the prediction it should send the array of data that updates the new value of tracked data every second to the Prediction Subsystem. Now the output of the Target Tracker Subsystem in APPENDIX B is not sufficient to meet this requirement. The Prediction Subsystem needs only the position values and it needs seven previous values in each dimension every second. By modifying the code in APPENDIX B, the first seven outputs are collected and send out in form of array in the same time. When the newest simulated data is produced it will update the value in the array as shown in APPENDIX C.

The Ada code in APPENDIX C shows the output of simulated data, but before running the Target Tracker Subsystem in the transputer network the only thing has to be modified is the output. In stead of printing out, just send these data to Prediction Subsystem using transputer communication links.

D. TARGET PREDICTION SUBSYSTEM

When the Tracker Subsystem keeps tracking the target and updates every one second, we want at least seven target's previous positions $(x_0,y_0,z_0),(x_1,y_1,z_1),...,(x_6,y_6,z_6)$ to predict the future position of the target by using the Least Square Orthogonal Polynomial to fit the path line of the target in the sense that the sum of the square of the distance from the curve orthogonal to each plane position is a minimum.

The special case of the estimation of linear parameters uses a linear combination of orthogonal polynomials to fit a smooth curve to a set of points with evenly spaced abscissas. A set of orthogonal polynomials commonly used for this purpose are the discrete Legendre Polynomial $O_{kn}(t)$, which satisfy the orthogonal relation

$$\sum_{t=0}^{n} O_{jn}(t)O_{kn}(t) = 0 \qquad (j \neq k)$$

One form of the discrete Legendre Polynomials for the evenly time-spaced abscissas t = 0,1,2,...,n is a set of $O_{kn}(t)$. The general formula for the kth-degree ($k \leq n$) discrete Legendre Polynomial is

$$O_{kn}(t) = \sum_{j=0}^{k} (-1)^j \binom{k}{j} \binom{k+j}{j} \frac{t^{(j)}}{n^{(j)}}$$

where the binomial coefficient

$$\binom{k}{j} = \frac{k!}{j!\ (k-j)!}$$

$$t^{(j)} = t(t-1)(t-2)...(t-j+1), \qquad t^{(0)} = 1$$

$$n^{(j)} = n(n-1)(n-2)...(n-j+1), \qquad n^{(0)} = 1$$

It is important to note that t has only integer value 0,1,2,...,n. So, a general form of $O_{kn}(t)$ are enumerated below

$$O_{0n}(t) = 1$$

$$O_{1n}(t) = 1 - 2\left(\frac{t}{n}\right)$$

$$O_{2n}(t) = 1 - 6\left(\frac{t}{n}\right) + 6\left(\frac{t}{n}\right)\left(\frac{t-1}{n-1}\right)$$

$$O_{3n}(t) = 1 - 12\left(\frac{t}{n}\right) + 30\left(\frac{t}{n}\right)\left(\frac{t-1}{n-1}\right) - 20\left(\frac{t}{n}\right)\left(\frac{t-1}{n-1}\right)\left(\frac{t-2}{n-2}\right)$$

By using seven previous positions (i.e. from time t = 0 to t = 6) of the target from the Tracker Subsystem, so t = 0,1,2,3,4,5,6 and n = 6. Therefore, the value of the first four Legendre Polynomial $O_{k6}(t)$ are given in table below.

1	O_{06}	O_{16}	O_{26}	O_{36}
[illegible]	1	1	1	[illegible]
1	1	4/6	[illegible]	-1
2	1	2/6	-3/5	-1
3	1	0	-4/5	0
[illegible]	1	-2/6	-3/5	1
5	1	-4/6	0	1
6	1	-1	1	-1

Another useful property of the discrete Legendre Polynomials is the following :

$$\sum_{t=0}^{n} O_{kn}(t) = \frac{(n+k+1)(n+k)^{(k)}}{(2k+1)(n)^{(k)}}$$

where $(n+k)^{(n)} = (n+k)(n+k-1)...(n+1)$

$(n)^{(k)} = n(n-1)(n-2)...(n-k+1)$

By setting $f_k(t) = O_{kn}(t)$ in expression above, we obtain linear combination of the Legendre Polynomial of the form

$$F(t) = a_0 O_{0n}(t) + a_1 O_{1n}(t) + a_2 O_{2n}(t) + ... + a_m O_{mn}(t)$$

And residuals, Q, are defined by the relation

$$r_t = F(t) - y_t \qquad (t = 0,n)$$

Therefore, the coefficient $a_0, a_1, ..., a_m$ are determined such that the sum of squared residuals $Q = \Sigma r_t^2 \equiv \Sigma [F(t) - y_t]^2$ is minimized

The normal equations obtained by setting the partial derivative $\partial Q/\partial a_0 = \partial Q/\partial a_1 = ... = \partial Q/\partial a_m = 0$, reduce to the form

$$\begin{vmatrix} \sum_{t=0}^{n} O_{0n}^2(t) & 0 & \cdots & 0 \\ 0 & \sum_{t=0}^{n} O_{1n}^2(t) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sum_{t=0}^{n} O_{mn}^2(t) \end{vmatrix} \begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{vmatrix} = \begin{vmatrix} \sum_{t=0}^{n} O_{0n}(t) y_t \\ \sum_{t=0}^{n} O_{1n}(t) y_t \\ \vdots \\ \sum_{t=0}^{n} O_{mn}(t) y_t \end{vmatrix}$$

Because of the orthogonal property the coefficient a_k(k=0,m) which produce the minimum Q are the solutions of the normal equations above ; these solutions are :

$$a_k = \frac{\sum_{t=0}^{n} O_{kn}(t) y_t}{\sum_{t=0}^{n} O_{kn}^2(t)} \qquad (k = 0,m)$$

To evaluate the unknown function at points other than the mesh points, each orthogonal polynomial of the linear combination, $F(t) = a_0O_{0n}(t) + a_1O_{1n}(t) + a_2O_{2n}(t) + ... + a_mO_{mn}(t)$ is replaced by its power of t representation, giving

$$F(t) = a_0 [\, 1 \,] + a_1 [\, 1 - 2 (\frac{t}{n}) \,] + a_2 [\, 1 - 6 (\frac{t}{n}) + 6 \frac{t\,(t-1)}{n\,(n-1)} \,] +$$

$$.... + a_m [\, 1 - \binom{m}{1}\binom{m+1}{1} (\frac{t}{n}) + \binom{m}{2}\binom{m+2}{2} \frac{t\,(t-1)}{n\,(n-1)} + ... \,]$$

Therefore, the time dependent function that approximate the path line of the target separately in three dimension is introduced, X(t), Y(t) and Z(t).

The Ada code in APPENDIX D performs the Prediction Subsystem. This program requires seven inputs of position value in each dimension which come from the Target Tracker Subsystem via transputer communication link and produces the orthogonal coefficients. Function ORTHOGONAL performs the numeric calculation of orthogonal Polynomial and function COEFF_CAL performs the calculation of the coefficient a_0, a_1, and a_2 that satisfied the sum of residuals square is minimized. Now we can fit the path line of the target with the Polynomial of order 2 and can predict the future positions of the target at anytime t.

E. BALLISTIC INTERCEPTION SUBSYSTEM

The predicted path line of the target is sent from the Target Prediction Subsystem in term of the position function depend on time in three dimension are

$$X(t) = a_0 + a_1 t + a_2 t^2$$

$$Y(t) = b_0 + b_1 t + b_2 t^2$$

$$Z(t) = c_0 + c_1 t + c_2 t^2$$

The actual values which are sent through the transputer link are just the value of the coefficient (a_0,a_1,a_2), (b_0,b_1,b_2) and (c_0,c_1,c_2). The main objective of the Ballistic Interception Subsystem is to compute the interception time.

1. Distance to the target

Since the Interception Subsystem receives the coefficient values from the Prediction Subsystem, the first value is the constant coefficient of the polynomial, the second value is the coefficient of the polynomial first order and the third value is the coefficient of the polynomial second order. The distance in each direction at any time t is known by substituting value of actual time t in the form

$$X(t=t) = a_0 + a_1 t + a_2 t^2$$

$$Y(t=t) = b_0 + b_1 t + b_2 t^2$$

$$Z(t=t) = c_0 + c_1 t + c_2 t^2$$

And the overall actual distance to the target at time t is

$$D = \sqrt{X^2(t)+Y^2(t)+Z^2(t)}$$

2. Interception Time

The time calculation is simply that time equal to distance divided by speed. Also the interception time equal to the distance to the intercept point divided by speed of the ammunition. The interception time is the actual time when target was detected plus the time required for the bullet to go to interception point. So, the interception distance in each direction is known by substitute the value of t with $t = t+t_{tof}$

$$X(t=t+t_{tof}) = a_0 + a_1(t+t_{tof}) + a_2(t+t_{tof})^2$$

$$Y(t=t+t_{tof}) = b_0 + b_1(t+t_{tof}) + b_2(t+t_{tof})^2$$

$$Z(t=t+t_{tof}) = c_0 + c_1(t+t_{tof}) + c_2(t+t_{tof})^2$$

And the overall interception distance at time $t+t_{tof}$ to the target is

$$D = \sqrt{X^2(t+t_{tof})+Y^2(t+t_{tof})+Z^2(t+t_{tof})}$$

In this thesis assume that the speed of the ammunition is constant equal to 2000 m/sec, and also assume that the trajectory of the bullet is the straight line. Therefore, the interception time is known by interception distance divided by 2000.

F. HOT SPARE

1. Fault Tolerance

It is not just desirable, but often essential, to support both safety (guarantee of not happening) properties and reliability (guarantee of happening) properties. Since system elements may fail, it is important to support the tolerance of such failures in both safety and reliability objectives. Fault tolerance deals with handling faults by restoring either full or reduced capability. Faults may have been foreseen but are not desired or controlled.

They may occur in any combination at unpredictable times and may require quick recovery, particularly in the real-time Combat Systems. Even if fault avoidance techniques applied in the designed phase can reduce the probability of faults but never eradicate them. Consequently there is the need to tolerate hardware faults during run-time so as to continue execution and preserve data integrity.

a. Hardware fault tolerance

Hardware faults vary in scale and duration, from transient memory faults to the failure of multi-processor node. Recovery from faults may be based on reloading and restarting lost processes. As parts of the system fail it may become unable to satisfy all the requirements of the application. Techniques for *graceful degradation* may be very useful to ensure that critical activities do not fail. Graceful degradation deals generally with reduced capability. When the system has a graceful degradation capability, its downtime for repair is short, uninterrupted operation is longer, unavailability periods are short, and overall processing power is not seriously affected by failure. In a multi processor system, graceful degradation can be achieved by re-configuring the system: switching out the faulty hardware or software and switching in the assumed good hardware or software, or masking the fail item: not using faulty hardware or software.

b. Software error tolerance

Software errors may be the result of residual design faults. Ada was designed to reduce residual design errors by encouraging highly modular and structured software design through the use of functional decomposition, information hiding and

strong type checking. Nevertheless software errors will occur in Ada programs. As with faults, error must be tolerated during run-time. The main requirements are similar, namely to continue execution, to preserve data integrity and to prevent the propagation of erroneous results. Two main techniques have been developed for software error tolerance, where errors are assumed to arise from design mistake, *recovery blocks* (Randell 1975) and *N-Version programming* (Chen & Avizienis 1978). These techniques are concerned with sequential programs and so are not effected by the organization of distributed Ada programs as communicating sequential processes.

2. Small Tactical System Fault Tolerance

Any system using more than one processor can have a fault-tolerant feature. This thesis concerns only the hardware fault tolerance. In the designed Small Tactical System one of the T800 20 Mhz transputer in the network is used to support the tolerance of such failures in the system. There are two cases of system failure that can occur frequently, loss of communication and transputer failure.

a. Loss of communication

The network of transputer whose links are connected through two crossbar switches. The links can be removed and/or inserted while the system is running. When the link is removed, the system looks for another link between the transputers that need to communicate and sets the crossbar switches to facilitate the needed communication. Thus, the fixed communication links are able to be replaced with dynamically assigned links for direct communication between transputers. These crossbar switches are

controlled by a single transputer that takes communication requirements from each of the other transputers in the system.

b. Transputer Failure

If an entire transputer node is fail, then every task that was executing on that node will have to be restarted. For each task to be restarted, it is necessary to recover the data of that task from some other transputer node. In the designed network of Small Tactical System, when one of the transputers in the network fails, one of transputer in the network, the HOT SPARE, should reload the software of the failed transputer and performs the same operations as the failed transputer.

IV. SMALL TACTICAL SYSTEM DEVELOPMENT

A. GENERAL

This chapter describes how to design and develop an Ada system for multi-transputer network as an efficient set of communicating programs with respect the data communicated between the programs especially the set of floating point number. The goals are to develop the Ada programs that perform difference task and run separately on each processor, which sent the computed data to the other programs. Specific details concerning the use of occam harness for the set of programs on each processor, and configuration descriptions for the *iconf* tool in the Occam 2 toolset.

B. MULTI-TRANSPUTER SYSTEMS

The Small Tactical System can be modelled by placing the different Ada program represented the task of each subsystem onto each transputer of the designed transputer network. The general idea of multi-transputer systems should be considered.

The main issue in the design of multi-transputer systems is that of configuration: the allocation of processes to a network of interconnected processors. Currently, the only possible distribution is *static distribution;* there is no explicit support provided in the Occam 2 Toolset for the dynamic allocation of processes to processors.

Since an Ada program may be considered as a process, Ada programs may also be distributed using the configuration tools supplied with the Occam 2 Toolset.

To implement an application on a five transputer network, involves four logical steps: Source Compilation, Object linking, Configuring, Loading and running.

1. Source Compilation

a. Source of Occam Harness

For multi-transputer systems, the task of configuration is greatly simplified if a harness is supplied for each processor. A major advantage of this form of harness structure is that the program to be run on each transputer is given a clean channel interface; the parameters of each harness are channels which will eventually be mapped onto physical links. Since the network is connected to host transputer using only a single link, The PROJ0 program is allocated to the root processor, the PROJ1, PROJ2, PROJ3 and PROJ4 programs are allocated to the remote processors. The main harness is used to invoke PROJ0 program and deal with communication with the host. Source of the occam harness must have the extension ".occ".

b. Source of Ada Programs

Single Ada programs cannot be distributed across a network; all tasks in the program execute on the same transputer. However, tasks in independent Ada programs can communicate using transputer channels via the implementation defined package CHANNELS. It is important that all processes in a transputer system complete their application processing cleanly. This causes control to be return to the PC host operating system and allows the system to be re-run without re-booting the network of transputers. The application Ada programs must have the extension ".ada".

2. Object Linking

Prior to configuration, all object binaries for each processor must be fully linked together. Linking is accomplished using the *ilink* tool. The linker is given the name of any separate compiled object or library files and produce the single object file where can be loaded onto each transputer in the network.

3. Configuring

Configuration is achieved using the *iconf* tool which takes a configuration description and produces an object file suitable for booting into a network of transputers. The purpose of the configuration description is to allocate code to processors and map channels used in the programs to physical links. The configuration description reflects the physical interconnection of the processors in the network. The root transputer is a T800 25 Mhz which connected via a single link to a network of 2 T800 20 Mhz processors.

The configure uses the configuration description to determine the topology of the network by analyzing the allocation of channels to physical links. Processor 0 is assumed to be the root transputer of the network through which the network is booted. There must be a route via transputer links from the root transputer to all other processors in the network. The following points should be remembered:

- All code used in a configuration description must be fully linked. There can be no explicit or implicit references to libraries.

- Any legal occam code may be written under a PROCESSOR statement. However, all code within these statements must be compiled in the same error mode and for the same processor type as specified in the PROCESSOR statement.

- Configuration channels must be placed on an input link on one processor and an output link on another processor. Channels placed only once are called *dangling links;* the configure produces a warning if such channels are used. Note that the channels used for communication with the server are dangling links since they are placed only once on the root processor.

- The same separately compiled program may be run on any number of processors; one copy of the code exists in the configured code and this is loaded onto each processor which requires it.

The default file extension for configuration description is ".pgm", the source of the configuration description in this thesis is "main.pgm", provided in APPENDIX F. When configuration is completed a new file, containing a bootable version of the code for the whole network, will have been created. The file have the same name as the description source, but with a ".btl" extension. So, in this thesis it would be "main.btl". A configuration description file with the ".dsc" extension is also created for debugger.

In order to take advantage of fast internal memory, it is possible for the main harness on each processor to control allocation of workspaces using the PLACE statement. Occam arrays may be stored in the occam scalar or vector workspaces using this mechanism. The PLACE statements override the default action of the occam compiler which itself is dependent upon a compiler switch.

4. Loading and Running

For the multi-transputer systems, the host file server is used to boot and load a network of transputers. The configure adds bootstrap code to initialize each transputer and route code to the appropriate processor. A communication protocol exists between the root transputer and a target transputer network to direct the loading of code to the desired

place in each transputer. Provided the harness for each processor is structured in the same way as the default harness, a program distributed over a network can be restarted.

The sequence of operation on transputer network is shown in Figure 12.

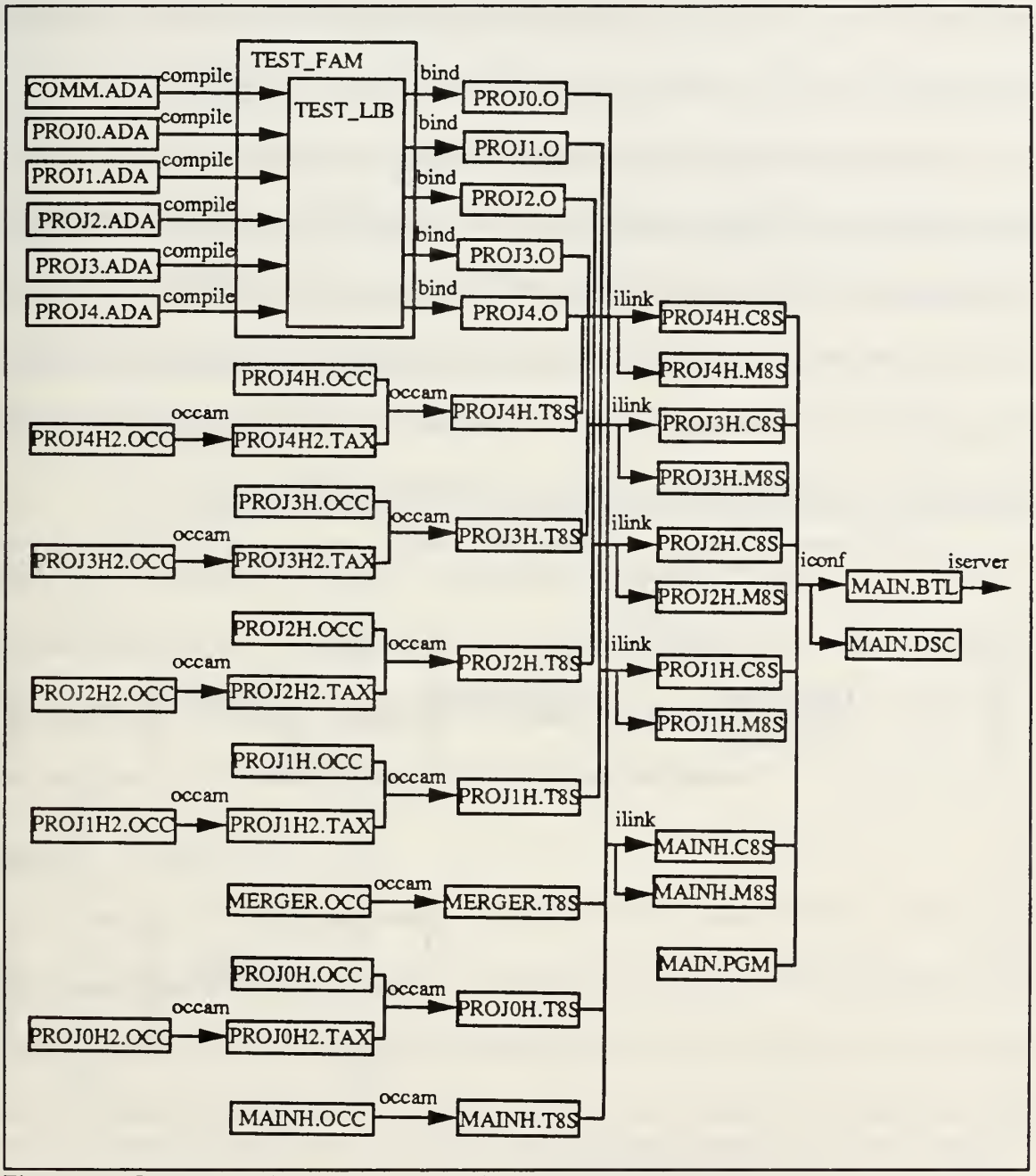

Figure 12: Sequence of operations on a five transputer network

C. TEST THE PERFORMANCE OF A 5 TRANSPUTER NETWORK

Before placing the Ada program that performs the task of each subsystem into each transputer in the network, the test Ada program has been made to ensure that all the performance in the network is correct. Each test Ada program should have the data communication exactly the same as the requirement of each subsystem.

The main requirement of data communication is to send and receive the three dimensional vector,(X,Y,Z), around the network. The specific type of data communication is declared in the package COMMON in the file common.ada in APPENDIX G. The idea of the test program is sending and receiving the three values of floating point number in term of vector communication around the transputer network. Each transputer executes the different Ada program. The test network will look like Figure 13.

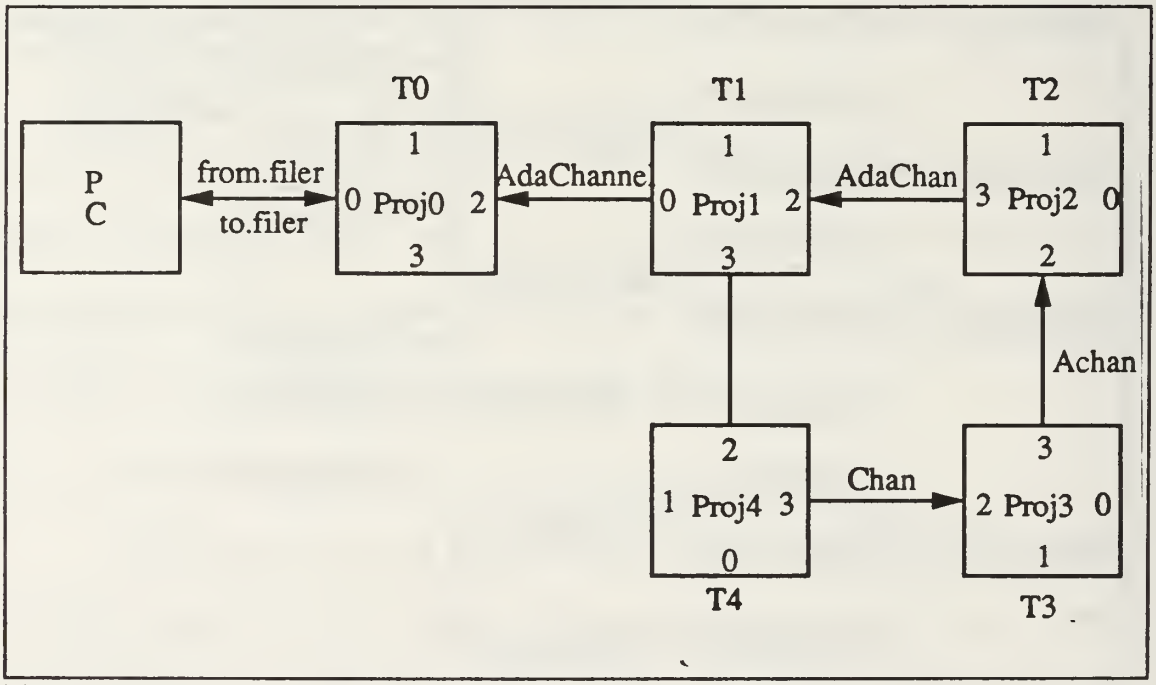

Figure 13: Test network of transputers

Each transputer performs different task in almost the same way. The main procedure of each program performs the production its own outputs and in the same time performs the communication in difference channel around the network. The communication links used in the network have to be defined the name of channel.

1. Transputer T_4

T_4 is the IMS T800 transputer in the IMS B003 evaluation board. In this board link3 of T_4 is connected to link2 of T_3. T_4 executes the proj4.ada which produces the output vector in three dimensions and sending these values to T_3 through the communication link3.out using the channel named "Chan".

2. Transputer T_3

T_3 is the IMS T800 transputer in the IMS B003 evaluation board. In this board link3 of T_3 is connected to link2 of T_2. T_3 executes the proj3.ada which produces its own outputs, receiving the output values of T_4 from the communication link2.in using the channel named "Chan", passing these values and sending its own outputs to T_2 through the same communication link3.out using channel named "Achan".

3. Transputer T_2

T_2 is the IMS T800 transputer in the IMS B003 evaluation board. In this board link3 of T_2 is connected to link2 of T_1. T_2 executes the proj2.ada which produces its own outputs, receiving the values from T_3 via communication link2.in using the channel named "Achan", passing the outputs of T_4, T_3 and sending its own outputs to T_1 through the same communication link3.out using the channel named "AdaChan".

4. Transputer T_1

T_1 is the IMS T800 transputer in the IMS B003 evaluation board. In this board link3 of T_1 is connected to link2 of T_4. Link0 of T_1 is connected to link2 of T_0 in the TMB08 TRAM motherboard. T_1 executes the proj1.ada which produce its own outputs, receiving the values from T_2 via communication link2.in using the channel named "AdaChan", passing the outputs of T_4, T_3, T_2 and sending its own outputs to T_0 through the same communication link0.out using the channel named "AdaChannel".

5. Transputer T_0

T_0 is the IMS T800 25 Mhz in the IMS B417 TRAM connected to PC using the TMB08 motherboard. Link2 of T_0 is connected to link0 of T_1 in the B003 evaluation board. T_0 executes the proj0.ada which produces its own outputs after receiving the values from T_1 via the communication link0.in using channel named "AdaChannel". T_0 performs as the host transputer communicating with the PC using the channel "from.filer" and "to.filer" to take care the print out of all the values passed from T_4, T_3, T_2, T_1 and its own outputs to the screen.

The Ada codes of proj0, proj1, proj2, proj3 and proj4 which run on T_0, T_1, T_2, T_3 and T_4 respectively are provided in APPENDIX G. This appendix also provides the output and package COMMON in file common.ada which is used to take care the specific type of data communication. In this case, the communicated data is the three dimensional array of floating point numbers.

D. SMALL TACTICAL SYSTEM

By the test performance of 5 transputer network in section C, the Small Tactical System can be developed. The idea of the Small Tactical System development is to place each subsystem into each transputer in the network as shown in Figure 14.

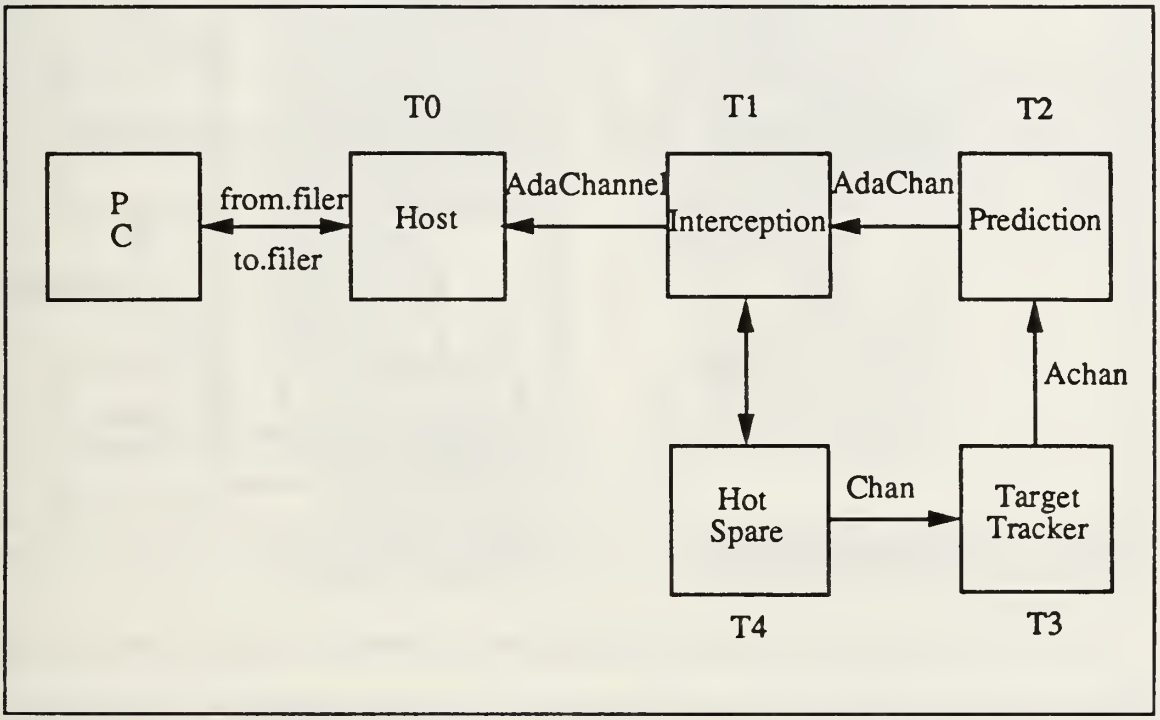

Figure 14: Modelling of Small Tactical System

Therefore, the performance of each transputer in the Small Tactical System network will be as the following

T_0 executes the Host program.

T_1 executes the Interception Subsystem program.

T_2 executes the Prediction Subsystem program.

T_3 executes the Target Tracker Subsystem Program.

T_4 executes the Hot Spare program.

Since the Interception Subsystem and the Hot Spare have not been developed yet, the Target Tracker Subsystem and Prediction Subsystem have been placed onto the transputer T_3 and T_2 respectively. The connection looks like Figure 15.

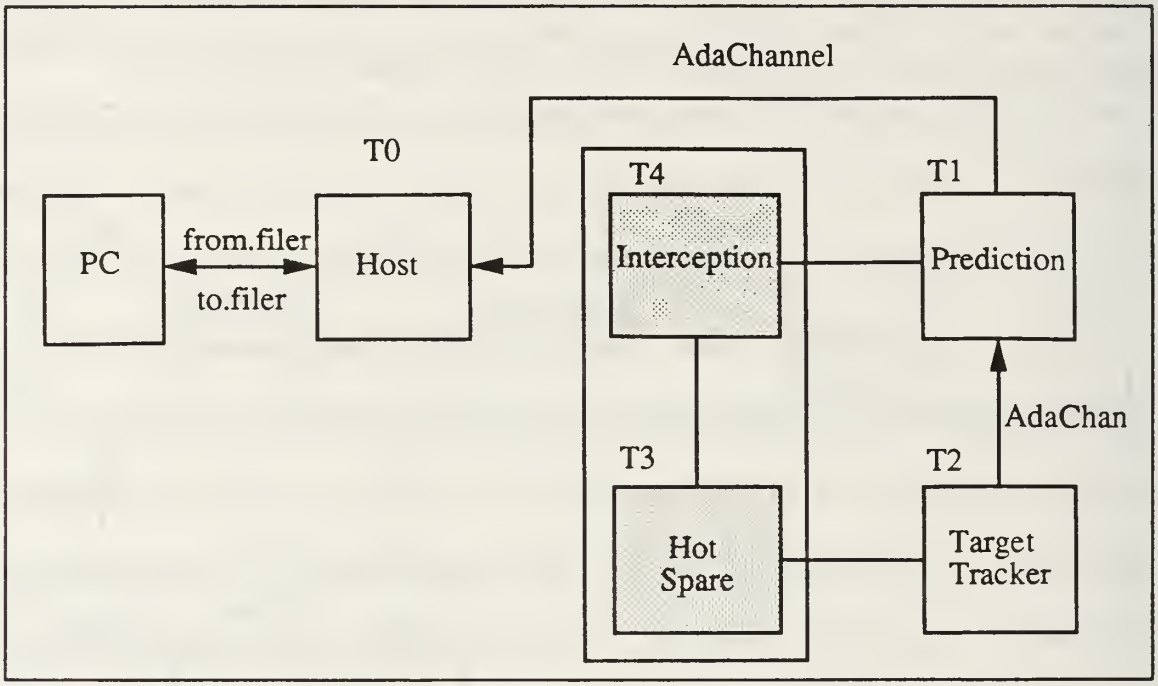

Figure 15: Connection Network

Due to changing the network, they might need to change a little bit in the programs and reconnected soft wires by connect link 2 of the T800 25 Mhz in the TMB08 board to link 1 of the second T800 20 Mhz in the B003 evaluation board. When the new connection has been made, the output of the check program looks like below

```
check 1.21
#  Part    rate  Mb    Bt [ Link0  Link1  Link2  Link3 ]
0  T800d -25   0.18   0 [ HOST   1:1    2:1     ...   ]
1  T2     -17  0.90   1 [   ...    0:1     ...   C004 ]
2  T800c -20   0.90   0 [   ...    0:2    3:3    4:2   ]
3  T800c -20   0.90   3 [   ...    ...    5:3    2:2   ]
4  T800c -20   0.90   2 [   ...    ...    2:3    5:2   ]
5  T800c -20   0.89   3 [   ...    ...    4:3    3:2   ]
```

The connection of this network is shown in Figure 16.

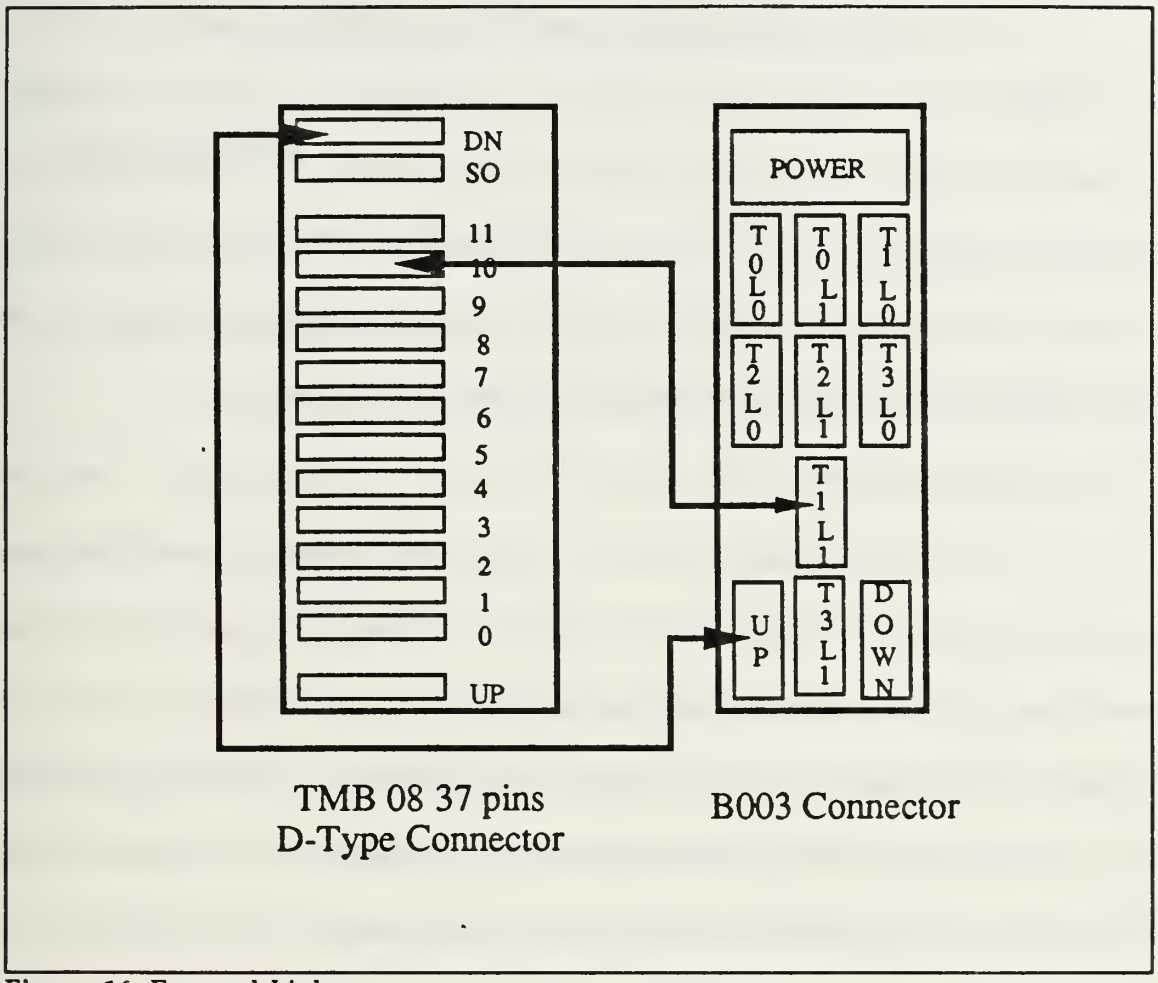

Figure 16: External Links

Now it looks like three transputers network. Using two T800 20 Mhz on the B003 evaluation board and one T800 25 Mhz TRAM to simulate the performance of Small Tactical System by placing the Ada program of Target Tracker Subsystem in APPENDIX C onto transputer T_2 and placed the Ada program of the Prediction Subsystem in APPENDIX D onto transputer T_1. The transputer T_0 takes care of the communication with host PC and prints out the output on the screen. The performance of each transputer in this network is described below

1. Transputer T_0

T_0 is the IMS T800 25 Mhz in the IMS B417 TRAM connected to PC using the TMB08 TRAM motherboard. Link2 of T_0 is connected to link1 of T_1 in the B003 evaluation board. T_0 executes the proj0.ada which receives the values from T_1 via communication link2.in using the channel named "AdaChannel". T_0 performs as the host transputer communicate with the PC using the channel "from.filer" and "to.filer" to take care the print out of all the final result of the Small Tactical System.

2. Transputer T_1

T_1 is the IMS T800 transputer in the IMS B003 evaluation board. Link1 of T_1 is connected to link2 of T_0 in the TMB08 TRAM motherboard. T_1 executes the proj1.ada which is the Prediction Subsystem, receiving the simulated tracked data from T_2 via communication link2.in using the channel named "AdaChan", and making the calculation using those values. It produces the coefficient values of Orthogonal Polynomial and sends to T_0 through the communication link1.out using the channel named "AdaChannel".

3. Transputer T_2

T_2 is the IMS T800 transputer in the IMS B003 evaluation board. Link3 of T_2 is connected to link2 of T_1. T_2 executes the proj4.ada which is the Target Tracker Subsystem. It produces the simulated tracked data vector in three dimensions, collects seven values of data by keep update these data every second and sending these values to T_1 through the communication link3.out using the channel named "AdaChan".

The code of three transputers network are provided in APPENDIX H.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

1. Ada on Transputers Network

a. Alsys-Ada Compiler

The Alsys-Ada Compiler includes a package named CHANNELS that implements communication on transputer links. It defines relevant data types, and procedures for channel I/O.

b. Occam Support

Ada code cannot be compiled to run on the transputer. Each Ada program must be supported with a harness of Occam. Other Occam programs (supplied with the Alsys distribution) handle merging error output, providing entry points to the code, defining the interface between the Ada procedures and the harnesses, and configuring the procedures for the transputer topology. The Ada and Occam objects are ultimately linked together into single module, regardless of the number of transputers. At run time, this module is decomposed by the Occam Toolset *iserver* and farmed out to the topology.

c. Issues

The limitations of running Ada programs on the transputer are hard to determine owing to the thin documentation. Trial and error must be employed. It is not yet clear what is the best method for passing data from Ada to Occam processes. Occam

provides elegant mechanisms for transmitting complex data structures. Ada has no such utility. Thus, the size of the Ada run-time code means that only one compiled Ada program can be run on a transputer. There is no facility for compiling multiple Ada objects with a single run-time system.

2. Formal Approach

This thesis is intended to provide the first stage or activity within the software development with Ada program on transputer by using the Alsys-ada compiler. Each subsystem runs the Ada program on each processor separately and at the same time each transputer sends the data through the communication links around the network. However, not all the goals could be refined as necessary. In the same way we do not expect the goal hierarchy to be complete. Future refinement goals, addition of new subgoals or reorganization of the hierarchy might be necessary as knowledge on details in gained in future stage of development.

One big problem that was found on developing the Small Tactical System is the Alsys-Ada Compilation System. In the complex Ada program, the data communication among the transputers network is difficult, especially the Ada programs that have the concurrency performance TASK by themselves. A lot of confusion in the protocols occurs when the transputers communicate different types of data through communication links.

The goal hierarchy is not yet completely defined, but the modelling of Small Tactical System shows that the commercial VLSI INMOS transputer can perform in the same way as the standard Navy's computer such as an AN/UYK-7 by running the software written in the standard Department of Defence programming language, Ada.

B. RECOMMENDATIONS

1. Future research

One of the requirement of the AEGIS combat system is that the system must have Fault Tolerance in the system itself. Since this thesis is not dealing with this part in depth, the software development which can detect all the faults diagnosed to one transputer can be reloaded in healthy transputers in order to continue functioning.

a. Commercial VLSI Microprocessor

The newest version of transputer, the INMOS T9000, will be available around Summer 1992. The T9000 is the implementation of a superscalar RISC architecture. It integrates a 32 bit processor, a 64 bit floating point unit, 16K bytes Cache, a communication processor and four links. The T9000 attains a peak performance of 200 MIPS, 25 MFLOPS and transmission rates of 800M Bits/sec at 50 Mhz. The T9000 is software compatible with the first generation of transputers and also provides an improved process and memory management. So, it is obvious that the T9000 has more features than the T805 transputer and they are software compatible.

b. Real-Time Operating System

One of the drawbacks in using MS-DOS operating system for developing the Small Tactical System is that it does not provide real-time mechanisms. To make the step from the prototype to an operating system, it is necessary to select one of the commercially available operating systems, which provide these mechanisms such as the TRANS-RTXC.

c. Classic-Ada

Classic-Ada is an extension to Ada that adds object-oriented features such as class and inheritance. It supports standard Ada in its entirety. The Classic-Ada software provides a toolset, most important of which is the Classic-Ada processor. The processor converts programs written in the Classic-Ada language into standard Ada. The resulting Ada programs can then be transferred to the PC and compiled using Alsys-Ada compilation system for transputers.

d. Man-Machine-Interface

The Man-Machine-Interface which is performed by the transputer T_0 in the transputer network will be one of the most complex modules within the Small Tactical System. Implementation of this interface using commercially available software tools should be considered. In order to illustrate the simplest output and easy to operate system, the application should contain software tools that include a text editor, menus, scroll bars, icons and command buttons.

2. Small Tactical System Future Versions

In the Aegis combat system, there are many sources of sensors that can provide the information of the target. The Tactical System also requires the information from the Navigational system to calculate the accuracy value of interception time. Based on the development of Ada version on transputer network implementation of Small Tactical System, we can provide some considerations on possible future version which can lead to development of the Navy's Aegis combat system update.

a. Radar Interface

The network can get the positional information from the raw video provided by radar system, format it and send it to the Target Tracker Subsystem. The Target Tracker Subsystem will keep track that target.

b. Link 11 Interface

The network also can get the positional information from some other source on communication system such as Link 11. There is a need of an interface that digitizes this information into positional data and sends it to Target Tracker Subsystem.

c. Weapon System Interface

One of the most important features of a combat system is the interface to a weapons system. The need for such an interface is certainly dependent on the type of ship. The interfaces should provide the operator with the capability to employ the weapon systems in their various operational modes.

APPENDIX A

Ada code that provides the comparison between Trapezoid integration and Simpson's integration. This program computes the aircraft location by using Trapezoidal rule and Simpson's rule of integration. It compares the results with the observed long-range radar. The aircraft assumed to fly back and forth between two points at altitude of 5000 m. The output of this lead to choose the better method used to simulate tracked data in the Target Tracker Subsystem.

```
-------------------------------------------------------------------------------------------------
-- Description  : This program computes the aircraft location by using Trapezoidal rule
--              : and Simpson's rule of integration. It compares the results with the
--              : observed long-range radars and computes error values for each method.
-------------------------------------------------------------------------------------------------
with TEXT_IO, GENERIC_ELEMENTARY_FUNCTIONS;
use TEXT_IO;

procedure PROJ is

   package MATH_FUNCT is new GENERIC_ELEMENTARY_FUNCTIONS(FLOAT);
   use MATH_FUNCT;

   package INTEGER_INOUT is new INTEGER_IO(INTEGER);
   use INTEGER_INOUT;
   package FLOAT_INOUT is new FLOAT_IO(FLOAT);
   use FLOAT_INOUT;

   type COMPONENT is (X, Y, Z);
   type VECTOR is array (COMPONENT) of FLOAT;
   type VEL_VALUES is array (0 .. 4) of VECTOR;
   DT    : constant FLOAT := 0.250;      -- delta time in seconds
   XP_T : VECTOR;                        -- X prediction using Trapezoidal integration
   XP_S : VECTOR;                        -- X prediction using Simpson's integration
   XL    : VECTOR;                       -- X as observed from the long-range radar
   T     : FLOAT := 0.0;                  -- elapsed time
   PI    : constant FLOAT := 3.1415_9265_3589_7932_3846_2643 ;
   VEL  : VEL_VALUES;                 -- calculated velocities

 procedure CALCULATE_VEL(T : in FLOAT; VEL : out VEL_VALUES) is
 --  This procedure CALCULATEs the VELocity vectors of the aircraft.
     C1 : constant FLOAT := 500.0 * PI / 6.0;
     C2 : constant FLOAT := PI / 60.0;
     ANGLE : FLOAT;
 begin
   for I in VEL_VALUES'RANGE loop
     ANGLE     :=   C2 * (T + DT * FLOAT(I));
     VEL(I) (X) := - C1 * SIN(ANGLE);
     VEL(I) (Y) :=   C1 * COS(ANGLE);
     VEL(I) (Z) :=    0.0;
   end loop;
 end CALCULATE_VEL;
```

```
function TRAPEZOID(XPOS:in VECTOR;VEL:in VEL_VALUES) return VECTOR is
  T : VECTOR;
begin
  for I in COMPONENT loop
    T(I) := 0.0;
    for J in VEL_VALUES'range loop
      T(I) := T(I) + VEL(J) (I);
    end loop;
    T(I):=T(I)-(VEL(VEL_VALUES'FIRST)(I)+VEL(VEL_VALUES'LAST)(I)) / 2.0;
    T(I) := XPOS(I) + DT * T(I);
  end loop;
  return T;
end TRAPEZOID;

function SIMPSON(XPOS:in VECTOR;VEL:in VEL_VALUES) return VECTOR is
  T : VECTOR;
begin
  for I in COMPONENT loop
  T(I):=(VEL(VEL_VALUES'FIRST)(I)+VEL(VEL_VALUES'LAST)(I));
    for J in VEL_VALUES'FIRST+1..VEL_VALUES'LAST-1 loop
      if (J MOD 2) = 1 then
        T(I) := T(I) + 4.0*VEL(J) (I);
      else
        T(I) := T(I) + 2.0*VEL(J) (I);
      end if;
    end loop;
    T(I) := XPOS(I) + DT * T(I) / 3.0;
  end loop;
  return T;
end SIMPSON;

function LOCATION(T : in FLOAT) return VECTOR is
  C1 : constant FLOAT := 5000.0;
  C2 : constant FLOAT := PI / 60.0;
  TT : VECTOR;
  ANGLE : FLOAT;
begin
  ANGLE := C2 * T;
  TT(X)   := C1 * COS(ANGLE);
  TT(Y)   := C1 * SIN(ANGLE);
  TT(Z)   := 4000.0;
  return TT;
end LOCATION;
```

```
function DIST(VEC_1 : in VECTOR; VEC_2 : in VECTOR) return FLOAT is
-- This function computes Euclidian DISTance between VEC_1 and VEC_2
  T : FLOAT := 0.0;
begin
  for I in VECTOR'RANGE loop
    T := T + (VEC_1(I) - VEC_2(I)) ** 2;
  end loop;
  return SQRT(T);
end DIST;

procedure PRINT(P : in VECTOR) is
-- This procedure PRINTs out values of one vector
begin
  PUT(P(X), FORE => 5, AFT => 4, EXP => 0);
  PUT(P(Y), FORE => 6, AFT => 4, EXP => 0);
  PUT(P(Z), FORE => 5, AFT => 0, EXP => 0);
end PRINT;

procedure PUT_HEADERS is
-- This procedure PUTs nice HEADERS before the data is printed out
begin
  PUT("_____________________________________________________________________________");
  NEW_LINE;
  PUT(" T  |");
  SET_COL(6);
  PUT("I. Trapezoidal integration");
  SET_COL(36);
  PUT("errors  |");
  SET_COL(46);
  PUT("II. Simpson's integration");
  SET_COL(76);
  PUT("errors");
  NEW_LINE;
  SET_COL(5);
  PUT(" | ");
  SET_COL(12);
  PUT("x         y         z   |         |");
  SET_COL(51);
  PUT("x         y         z   |");
  NEW_LINE;
  PUT("_____________________________________________________________________________");
  NEW_LINE;
end PUT_HEADERS;
```

```
begin -- main program

  XP_T(X) := 5000.0;
  XP_T(Y) := 0.0;
  XP_T(Z) := 4000.0;
  XP_S(X) := XP_T(X);
  XP_S(Y) := XP_T(Y);
  XP_S(Z) := XP_T(Z);
  NEW_LINE(2);
  PUT_HEADERS;
  for I in 1 .. 20 loop
    for J in 1..30 loop
      for K in 0..4 loop
        CALCULATE_VEL(T, VEL);
      end loop;
      T := T + 1.0;
      XP_T := TRAPEZOID(XP_T, VEL);
      XP_S := SIMPSON(XP_S,VEL);
    end loop;
    -- print out results at 30 seconds regular intervals
    XL := LOCATION(T);
    PUT(INTEGER(T), WIDTH => 3);
    PUT(" | ");
    PRINT(XP_T);
    PUT(" | ");
    PUT(DIST(XP_T, XL), FORE => 2, AFT => 4, EXP => 0);
    PUT(" | ");
    SET_COL(44);
    PRINT(XP_S);
    PUT(" |");
    PUT(DIST(XP_S, XL), FORE => 2, AFT => 4, EXP => 0);
    NEW_LINE;
  end loop;
end PROJ;
```

T	I. Trapezoidal integration			errors	II. Simpson's integration			errors
	x	y	z		x	y	z	
30	0.0713	4999.9282	4000.0	0.1013	-0.0003	5000.0004	4000.0	0.0005
60	-4999.8569	-0.0007	4000.0	0.1430	-5000.0004	-0.0003	4000.0	0.0004
90	0.0722	-4999.9287	4000.0	0.1014	0.0005	-5000.0000	4000.0	0.0005
120	4999.9990	0.0005	4000.0	0.0010	5000.0000	0.0010	4000.0	0.0001
150	0.0697	4999.9272	4000.0	0.1019	-0.0010	4999.9995	4000.0	0.0007
180	-4999.8564	-0.0023	4000.0	0.1435	-5000.0000	-0.0016	4000.0	0.0015
210	0.0740	-4999.9287	4000.0	0.1004	0.0013	-5000.0000	4000.0	0.0019
240	4999.9995	0.0021	4000.0	0.0006	4999.9980	0.0013	4000.0	0.0020
270	0.0684	4999.9277	4000.0	0.0996	-0.0033	4999.9990	4000.0	0.0033
300	-4999.8574	-0.0035	4000.0	0.1425	-5000.0004	-0.0028	4000.0	0.0007
330	0.0732	-4999.9287	4000.0	0.0975	0.0017	-4999.9995	4000.0	0.0048
360	4999.9985	0.0018	4000.0	0.0022	4999.9995	0.0024	4000.0	0.0022
390	0.0672	4999.9272	4000.0	0.1014	-0.0038	4999.9995	4000.0	0.0006
420	-4999.8574	-0.0043	4000.0	0.1425	-5000.0000	-0.0039	4000.0	0.0026
450	0.0742	-4999.9291	4000.0	0.1023	0.0028	-5000.0009	4000.0	0.0026
480	4999.9995	0.0026	4000.0	0.0009	4999.9995	0.0025	4000.0	0.0010
510	0.0671	4999.9277	4000.0	0.1033	-0.0044	4999.9990	4000.0	0.0024
540	-4999.8569	-0.0043	4000.0	0.1431	-5000.0009	-0.0037	4000.0	0.0035
570	0.0756	-4999.9291	4000.0	0.1010	0.0030	-4999.9990	4000.0	0.0010
600	4999.9995	0.0041	4000.0	0.0026	4999.9985	0.0051	4000.0	0.0021

APPENDIX B

Ada code for the Target Tracker Subsystem. The objective of this program is to show the simulate data which is sent to the Prediction Subsystem. The actual required of the Prediction subsystem is only the positions of the target in X,Y and Z direction but output of this program also shows the velocities in three dimension. Assume that when radar starts to track the target, it approaches at some acceleration until reaching its maximum speed, and then it approaches with constant velocities. In this case assume the target is a missile whose maximum speed is Mach 3. The initial position that is assumed to be the maximum tracking range of the radar (assume to be 35 Km.). The tracked data simulation is terminated when the target is too close that the track radar cannot longer keep track it anymore.

The first three column of the output is the simulated tracked data that will send to the Prediction Subsystem. Since the Prediction Subsystem needs at least seven positions of the tracked data this code will be modified before put in the network of transputers.

```
with TEXT_IO, CALENDAR, GENERIC_ELEMENTARY_FUNCTIONS;
use  TEXT_IO, CALENDAR;

procedure PROJ is

   package MATH_FUNCT is new GENERIC_ELEMENTARY_FUNCTIONS(FLOAT);
   use MATH_FUNCT;

   package FLOAT_INOUT is new FLOAT_IO(FLOAT);
   use FLOAT_INOUT;

   -- Type identifications
   type COMPONENT is (X,Y,Z);
   type VECTOR is array (COMPONENT) of FLOAT;
   type VELOCITIES is array (0..4) of VECTOR;

   DELTA_TIME : constant FLOAT := 0.25;  -- delta time t = 1/4 seconds

   NO_TARGET : BOOLEAN := FALSE;
   CURRENT     : constant INTEGER := 4;
   LINT_SEC    : INTEGER;
   INTERVAL    : DAY_DURATION := 1.0;
   DISP_TIME   : DAY_DURATION := 0.0;
   POSITION    : VECTOR := (27000.0,22000.0,5000.0);
   INT_VEL     : VECTOR := (230.0,180.0,25.0);
   VELOCITY    : VELOCITIES;

   function "+"(LEFT, RIGHT : in VECTOR) return VECTOR is
   ------------------------------------------------------------------------
   -- This function is written to handle VECTOR addition.
   ------------------------------------------------------------------------
     TEMP : VECTOR;
   begin
     TEMP(X) := LEFT(X) + RIGHT(X);
     TEMP(Y) := LEFT(Y) + RIGHT(Y);
     TEMP(Z) := LEFT(Z) + RIGHT(Z);
     return TEMP;
   end "+";
```

```
    function "-"(LEFT, RIGHT : in VECTOR) return VECTOR is
  ------------------------------------------------------------------------------
  -- This function is written to handle VECTOR subtraction.
  ------------------------------------------------------------------------------
    TEMP : VECTOR;
  begin
    TEMP(X) := LEFT(X) - RIGHT(X);
    TEMP(Y) := LEFT(Y) - RIGHT(Y);
    TEMP(Z) := LEFT(Z) - RIGHT(Z);
    return TEMP;
  end "-";

  function SIMPSON(XPOS:in VECTOR; VEL:in VELOCITIES) return VECTOR is
  ------------------------------------------------------------------------------------------------------
  --  This function performs numeric integration using Simpson's rule and return a
  --  position vector giving a set of sample VELOCITIES.
  ------------------------------------------------------------------------------------------------------
     T : VECTOR;
     I : COMPONENT;
     J : INTEGER;
  begin
     for I in COMPONENT loop
        T(I) := (VEL(VELOCITIES'FIRST) (I) + VEL(VELOCITIES'LAST) (I));
        for J in VELOCITIES'FIRST+1..VELOCITIES'LAST-1 loop
           if (J MOD 2) = 1 then
              T(I) := T(I) + 4.0*VEL(J) (I);
           else
              T(I) := T(I) + 2.0*VEL(J) (I);
           end if;
        end loop;
        T(I) :=  DELTA_TIME * T(I) / 3.0;
     end loop;
     return T;
  end SIMPSON;


    package ATOD is
  ------------------------------------------------------------------------------
  -- This package is used to maintain the velocity values for the previous
  -- second at 1/4 second intervals (five values). It has one function
  -- which returns an array of five vector. The task is written to handle
  -- concurrent processing of ACCELEROMETER.
  ------------------------------------------------------------------------------
```

```
    procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR);
    procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES);
    task ACCELEROMETER is
        entry START;
    end ACCELEROMETER;
end ATOD;

package body ATOD is
    VEL : VELOCITIES := (others => (others => 0.0));

    procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR) is
        I : INTEGER;
    begin
        for I in VELOCITIES'RANGE loop
            VEL(I) := FIRST_VEL;
        end loop;
    end INITIALIZE_VELOCITY;

    procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES) is
    begin
        NEW_VEL := VEL;
    end GET_VELOCITIES;

    task body ACCELEROMETER is
        use CALENDAR;
        INTERVAL  : constant DURATION := 0.25;
        DISP_TIME : DURATION := 0.0;
        LINT_SEC  : INTEGER  := 0;
    begin
        accept START do
        null;
        end START;

        LINT_SEC := INTEGER(SECONDS(CLOCK));
        DISP_TIME := DURATION(LINT_SEC);
        while DISP_TIME < SECONDS(CLOCK) loop
            DISP_TIME := DISP_TIME + INTERVAL;
        end loop;
        loop
            delay DISP_TIME - SECONDS(CLOCK);
            for I in VELOCITIES'FIRST..VELOCITIES'LAST-1 loop
                VEL(I) := VEL(I+1);
            end loop;
```

```
            VEL(VELOCITIES'LAST) := VEL(VELOCITIES'LAST)+
                                    (0.012,0.0098,0.00275);
            exit when VEL(4)(X) > 700.0;
         end loop;
      end ACCELEROMETER;
   end ATOD;

   procedure PUT_POSITION_VELOCITY (XP: in VECTOR; VEL: in VECTOR) is
   begin
      SET_COL(2);
      PUT(XP(X), FORE => 6, AFT => 4, EXP => 0);
      PUT(" ");
      PUT(XP(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT(" ");
      PUT(XP(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT(" ");
      PUT(VEL(X), FORE => 6, AFT => 4, EXP => 0);
      PUT(" ");
      PUT(VEL(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT(" ");
      PUT(VEL(Z), FORE => 6, AFT => 4, EXP => 0);
      NEW_LINE;
   end PUT_POSITION_VELOCITY;

begin  -- main program

   ATOD.INITIALIZE_VELOCITY(INT_VEL);
   LINT_SEC := INTEGER(SECONDS(CLOCK));
   DISP_TIME := DURATION(LINT_SEC) + 0.8;
   ATOD.ACCELEROMETER.START;
   while NO_TARGET = FALSE loop
      if POSITION(X) > 0.0 then
         delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
         ATOD.GET_VELOCITIES(VELOCITY);
         PUT_POSITION_VELOCITY(POSITION,VELOCITY(CURRENT));
         POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
         DISP_TIME := DISP_TIME + INTERVAL;
      else
         NO_TARGET := TRUE;
      end if;
   end loop;
end PROJ;
```

Output of this program shows that the first 12 second the target approaches with some acceleration and when reaching its maximum speed it approaches with constant velocities. The first 3 columns are the positions and the last 3 column are the velocities.

27000.0000	22000.0000	5000.0000	230.0000	180.0000	25.0000
26770.0000	21820.0000	4975.0000	267.6113	210.7207	33.6252
26502.4121	21609.2988	4941.3803	315.5249	249.8562	44.6130
26186.9101	21359.4628	4896.7729	363.4386	288.9918	55.6009
25823.4960	21070.4902	4841.1777	411.3523	328.1274	66.5851
25412.1679	20742.3828	4774.5981	459.2659	367.2630	77.5577
24952.9257	20375.1386	4697.0458	507.1916	406.4084	88.5330
24445.7578	19968.7500	4608.5180	555.1789	445.5146	99.4974
23890.6035	19523.2558	4509.0258	603.2025	484.6404	110.4673
23287.4257	19038.6347	4398.5639	651.2020	523.7832	121.4316
22636.2480	18514.8710	4277.1376	699.2015	563.0112	132.3960
21937.0703	17951.8789	4144.7470	700.0071	563.6696	132.5800
21237.0878	17388.2285	4012.1726	700.0071	563.6696	132.5800
20537.1054	16824.5781	3879.5981	700.0071	563.6696	132.5800
19837.1230	16260.9277	3747.0236	700.0071	563.6696	132.5800
19137.1406	15697.2773	3614.4492	700.0071	563.6696	132.5800
18437.1582	15133.6269	3481.8747	700.0071	563.6696	132.5800
17737.1757	14569.9765	3349.3002	700.0071	563.6696	132.5800
17037.1933	14006.3261	3216.7258	700.0071	563.6696	132.5800
16337.2099	13442.6757	3084.1513	700.0071	563.6696	132.5800
15637.2265	12879.0253	2951.5769	700.0071	563.6696	132.5800
14937.2431	12315.3750	2819.0024	700.0071	563.6696	132.5800
14237.2597	11751.7246	2686.4279	700.0071	563.6696	132.5800
13537.2763	11188.0742	2553.8535	700.0071	563.6696	132.5800
12837.2929	10624.4238	2421.2790	700.0071	563.6696	132.5800
12137.3095	10060.7734	2288.7045	700.0071	563.6696	132.5800
11437.3261	9497.1230	2156.1301	700.0071	563.6696	132.5800
10737.3427	8933.4726	2023.5555	700.0071	563.6696	132.5800
10037.3593	8369.8222	1890.9809	700.0071	563.6696	132.5800
9337.3759	7806.1723	1758.4063	700.0071	563.6696	132.5800
8637.3925	7242.5224	1625.8317	700.0071	563.6696	132.5800
7937.4096	6678.8725	1493.2572	700.0071	563.6696	132.5800
7237.4267	6115.2226	1360.6826	700.0071	563.6696	132.5800
6537.4438	5551.5727	1228.1080	700.0071	563.6696	132.5800

APPENDIX C

Since the Prediction Subsystem requires at least seven position values in each dimension. The Ada code in this appendix is the modified code from APPENDIX B that prepare the output ready to send to the Prediction Subsystem in term of array 3*7 of floating point numbers. The output of this program is $(X_0,Y_0,Z_0),(X_1,Y_1,Z_1),...,(X_6,Y_6,Z_6)$.

```
-- File: proj.ada

with TEXT_IO, CALENDAR, GENERIC_ELEMENTARY_FUNCTIONS;
use  TEXT_IO, CALENDAR;


procedure PROJ is

   package MATH_FUNCT is new GENERIC_ELEMENTARY_FUNCTIONS(FLOAT);
   use MATH_FUNCT;

   package FLOAT_INOUT is new FLOAT_IO(FLOAT);
   use FLOAT_INOUT;

   -- Type identifications
   type COMPONENT is (X,Y,Z);
   type VECTOR is array (COMPONENT) of FLOAT;
   type VELOCITIES is array (0..4) of VECTOR;
   type ARY_7 is array (0..6) of VECTOR;

   DELTA_TIME : constant FLOAT := 0.25;  -- delta time t = 1/4 seconds
   NO_TARGET  : BOOLEAN := FALSE;
   CURRENT    : constant INTEGER := 4;
   LINT_SEC   : INTEGER;
   INTERVAL   : DAY_DURATION := 1.0;
   DISP_TIME  : DAY_DURATION := 0.0;
   POSITION   : VECTOR := (27000.0,22000.0,5000.0);
   P,P0,P1,P2,P3,P4,P5,P6 : VECTOR := (0.0,0.0,0.0);
   INT_VEL    : VECTOR := (230.0,180.0,25.0);
   VELOCITY   : VELOCITIES;
   B : ARY_7 ;

   function "+"(LEFT, RIGHT : in VECTOR) return VECTOR is
   ------------------------------------------------------------------------
   -- This function is written to handle VECTOR addition.
   ------------------------------------------------------------------------
      TEMP : VECTOR;
   begin
      TEMP(X) := LEFT(X) + RIGHT(X);
      TEMP(Y) := LEFT(Y) + RIGHT(Y);
      TEMP(Z) := LEFT(Z) + RIGHT(Z);
      return TEMP;
   end "+";
```

```
function "-"(LEFT, RIGHT : in VECTOR) return VECTOR is
-------------------------------------------------------------------------
-- This function is written to handle VECTOR subtraction.
-------------------------------------------------------------------------
   TEMP : VECTOR;
begin
   TEMP(X) := LEFT(X) - RIGHT(X);
   TEMP(Y) := LEFT(Y) - RIGHT(Y);
   TEMP(Z) := LEFT(Z) - RIGHT(Z);
   return TEMP;
end "-";

function SIMPSON(XPOS: in VECTOR; VEL: in VELOCITIES) return VECTOR is
-------------------------------------------------------------------------------------------------------
-- This function performs numeric integration using SIMPSONS rule and return a
-- position giving a set of sample VELOCITIES. and the DELTA_TIME between
-- those velocities.
-------------------------------------------------------------------------------------------------------
   T : VECTOR;
   I : COMPONENT;
   J : INTEGER;
begin
   for I in COMPONENT loop
      T(I) := (VEL(VELOCITIES'FIRST) (I) + VEL(VELOCITIES'LAST) (I));
      for J in VELOCITIES'FIRST+1..VELOCITIES'LAST-1 loop
         if (J MOD 2) = 1 then
            T(I) := T(I) + 4.0*VEL(J) (I);
         else
            T(I) := T(I) + 2.0*VEL(J) (I);
         end if;
      end loop;
      T(I) :=  DELTA_TIME * T(I) / 3.0;
   end loop;
   return T;
end SIMPSON;


  package ATOD is
-------------------------------------------------------------------------------------------------------
-- This package is used to maintain the velocity values for the previous second at 1/4
-- second intervals (five values). It has one function which returns an array of five
-- vector. The task is written to handle concurrent processing of ACCELEROMETER.
-------------------------------------------------------------------------------------------------------
```

```
   procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR);
   procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES);
   task ACCELEROMETER is
      entry START;
   end ACCELEROMETER;
end ATOD;

package body ATOD is
   VEL : VELOCITIES := (others => (others => 0.0));

   procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR) is
      I : INTEGER;
   begin
      for I in VELOCITIES'RANGE loop
         VEL(I) := FIRST_VEL;
      end loop;
   end INITIALIZE_VELOCITY;

   procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES) is
   begin
      NEW_VEL := VEL;
   end GET_VELOCITIES;

   task body ACCELEROMETER is
      use CALENDAR;
      INTERVAL  : constant DURATION := 0.25;
      DISP_TIME : DURATION := 0.0;
      LINT_SEC  : INTEGER  := 0;
   begin
      accept START do
      null;
      end START;

      LINT_SEC := INTEGER(SECONDS(CLOCK));
      DISP_TIME := DURATION(LINT_SEC);
      while DISP_TIME < SECONDS(CLOCK) loop
         DISP_TIME := DISP_TIME + INTERVAL;
      end loop;
      loop
         delay DISP_TIME - SECONDS(CLOCK);
         for I in VELOCITIES'FIRST..VELOCITIES'LAST-1 loop
            VEL(I) := VEL(I+1);
         end loop;
```

```
            VEL(VELOCITIES'LAST) := VEL(VELOCITIES'LAST) +
                                        (0.012,0.0098,0.00275);
            exit when VEL(4)(X) > 700.0;
        end loop;
    end ACCELEROMETER;

end ATOD;


begin  -- main program

    ATOD.INITIALIZE_VELOCITY(INT_VEL);
    LINT_SEC := INTEGER(SECONDS(CLOCK));
    DISP_TIME := DURATION(LINT_SEC) + 0.8;
    ATOD.ACCELEROMETER.START;

    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
    P0(X) := POSITION(X);
    P0(Y) := POSITION(Y);
    P0(Z) := POSITION(Z);
    B(0)(X) := P0(X);
    B(0)(Y) := P0(Y);
    B(0)(Z) := P0(Z);

    POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
    DISP_TIME := DISP_TIME + INTERVAL;
    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
    P1(X) := POSITION(X);
    P1(Y) := POSITION(Y);
    P1(Z) := POSITION(Z);
    B(1)(X) := P1(X);
    B(1)(Y) := P1(Y);
    B(1)(Z) := P1(Z);

    POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
    DISP_TIME := DISP_TIME + INTERVAL;
    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
    P2(X) := POSITION(X);
    P2(Y) := POSITION(Y);
    P2(Z) := POSITION(Z);
```

```
B(2)(X) := P2(X);
B(2)(Y) := P2(Y);
B(2)(Z) := P2(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P3(X) := POSITION(X);
P3(Y) := POSITION(Y);
P3(Z) := POSITION(Z);
B(3)(X) := P3(X);
B(3)(Y) := P3(Y);
B(3)(Z) := P3(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P4(X) := POSITION(X);
P4(Y) := POSITION(Y);
P4(Z) := POSITION(Z);
B(4)(X) := P4(X);
B(4)(Y) := P4(Y);
B(4)(Z) := P4(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P5(X) := POSITION(X);
P5(Y) := POSITION(Y);
P5(Z) := POSITION(Z);
B(5)(X) := P5(X);
B(5)(Y) := P5(Y);
B(5)(Z) := P5(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P6(X) := POSITION(X);
P6(Y) := POSITION(Y);
```

```
P6(Z) := POSITION(Z);
B(6)(X) := P6(X);
B(6)(Y) := P6(Y);
B(6)(Z) := P6(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
while NO_TARGET = FALSE loop
   if POSITION(X) > 0.0 then
      PUT (B(0)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(1)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(2)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(3)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(4)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(5)(X), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(6)(X), FORE => 6, AFT => 4, EXP => 0);
      NEW_LINE;

      PUT (B(0)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(1)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(2)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(3)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(4)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(5)(Y), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(6)(Y), FORE => 6, AFT => 4, EXP => 0);
      NEW_LINE;

      PUT (B(0)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(1)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(2)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(3)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(4)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(5)(Z), FORE => 6, AFT => 4, EXP => 0);
      PUT (B(6)(Z), FORE => 6, AFT => 4, EXP => 0);
      NEW_LINE(3);

      delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
      ATOD.GET_VELOCITIES(VELOCITY);
      P(X) := POSITION(X);
      P(Y) := POSITION(Y);
      P(Z) := POSITION(Z);
      POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
      DISP_TIME := DISP_TIME + INTERVAL;
```

```
            P0 := P1;
            P1 := P2;
            P2 := P3;
            P3 := P4;
            P4 := P5;
            P5 := P6;
            P6 := P;

            B(0)(X)  := P0(X);
            B(1)(X)  := P1(X);
            B(2)(X)  := P2(X);
            B(3)(X)  := P3(X);
            B(4)(X)  := P4(X);
            B(5)(X)  := P5(X);
            B(6)(X)  := P6(X);

            B(0)(Y)  := P0(Y);
            B(1)(Y)  := P1(Y);
            B(2)(Y)  := P2(Y);
            B(3)(Y)  := P3(Y);
            B(4)(Y)  := P4(Y);
            B(5)(Y)  := P5(Y);
            B(6)(Y)  := P6(Y);

            B(0)(Z)  := P0(Z);
            B(1)(Z)  := P1(Z);
            B(2)(Z)  := P2(Z);
            B(3)(Z)  := P3(Z);
            B(4)(Z)  := P4(Z);
            B(5)(Z)  := P5(Z);
            B(6)(Z)  := P6(Z);

        else
            NO_TARGET := TRUE;
        end if;
    end loop;

end PROJ;
```

APPENDIX D

Ada code that provides the performance of Prediction SubSystem. It needs at least seven values of position vector and computes the Orthogonal Polynomial coefficient.

```
-- File: proj1.ada

with TEXT_IO, COMMON, CHANNELS;
use  TEXT_IO, COMMON;

procedure PROJ1 is

   B : ARY_7;
   RESULT : VEC_3;
   TOTAL  : VEC_3;

   C : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   D : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel used for communication with the program producing the stream.
   -- Input channels 0 and 1 are reserved for use by the run-time system.

    package FLOAT_INOUT is new FLOAT_IO(FLOAT);
    use FLOAT_INOUT;

    -- type identification
    type COMPONENT is (X0,X1,X2,X3,X4,X5,X6);
    type VECTOR is array (COMPONENT) of FLOAT;
    type POSITIONS is array (0..3) of FLOAT;
    type VEC is array (0..2) of FLOAT;

    -- orthogonal constant
    CONST0 : VECTOR := (1.0,1.0,1.0,1.0,1.0,1.0,1.0);
    CONST1 : VECTOR := (1.0,0.666,0.333,0.0,-0.333,-0.666,-1.0);
    CONST2 : VECTOR := (1.0,0.0,-0.6,-0.8,-0.6,0.0,1.0);
    CONST3 : VECTOR := (1.0,-1.0,-1.0,0.0,1.0,1.0,-1.0);
    T    : VECTOR := (0.0,0.0,0.0,0.0,0.0,0.0,0.0);
    POS  : POSITIONS := (0.0,0.0,0.0,0.0);
    COEF : VEC;

    function VECTOR_SUM (X : VECTOR) return FLOAT is
       SUM : FLOAT := 0.0;
    begin
       for I in X'RANGE loop
          SUM := SUM + X(I);
       end loop;
       return SUM;
    end VECTOR_SUM;
```

```
function VECTOR_SUM_SQUARE (X : VECTOR) return FLOAT is
   SUM_SQUARE : FLOAT := 0.0;
begin
   for I in X'RANGE loop
      SUM_SQUARE := SUM_SQUARE + (X(I)**2);
   end loop;
   return SUM_SQUARE;
end VECTOR_SUM_SQUARE;


function ORTHOGONAL (POS_IN : VECTOR) return POSITIONS is
   TEMP0,TEMP1,TEMP2,TEMP3 : VECTOR;
   POS_OUT0,POS_OUT1,POS_OUT2,POS_OUT3 : FLOAT;
   ORTHO : POSITIONS;
begin
   for I in COMPONENT loop
      TEMP0(I) := POS_IN(I) * CONST0(I);
      TEMP1(I) := POS_IN(I) * CONST1(I);
      TEMP2(I) := POS_IN(I) * CONST2(I);
      TEMP3(I) := POS_IN(I) * CONST3(I);
   end loop;
   POS_OUT0 := VECTOR_SUM(TEMP0)/VECTOR_SUM_SQUARE(CONST0);
   POS_OUT1 := VECTOR_SUM(TEMP1)/VECTOR_SUM_SQUARE(CONST1);
   POS_OUT2 := VECTOR_SUM(TEMP2)/VECTOR_SUM_SQUARE(CONST2);
   POS_OUT3 := VECTOR_SUM(TEMP3)/VECTOR_SUM_SQUARE(CONST3);
   ORTHO := (POS_OUT0,POS_OUT1,POS_OUT2,POS_OUT3);
   return ORTHO;
end ORTHOGONAL;


function COEFF_CAL (A : in POSITIONS) return VEC is
   a0,a1,a2 : FLOAT;
   V : VEC;
begin
   a0 := A(0)+A(1)+A(2)+A(3);
   a1 := (-A(1)*(0.3333))+(A(2)*(-1.2))+(-A(3)*(0.6667));
   a2 := (A(2)*(0.2000))+(A(3)*(1.500));
   V := (a0,a1,a2);
   return V;
end COEFF_CAL;
```

```
begin
   loop
      VECTOR_IO.READ (C, RESULT);
      VECTOR_IO.WRITE (D, RESULT);
      exit when RESULT(0) < 0.0;
      ARRAY_IO.READ (C, B);
      T(X0) := FLOAT(B(0)(0));
      T(X1) := FLOAT(B(1)(0));
      T(X2) := FLOAT(B(2)(0));
      T(X3) := FLOAT(B(3)(0));
      T(X4) := FLOAT(B(4)(0));
      T(X5) := FLOAT(B(5)(0));
      T(X6) := FLOAT(B(6)(0));
      POS := ORTHOGONAL(T);
      COEF := COEFF_CAL(POS);
      TOTAL(0) := FLO_6(COEF(0));
      TOTAL(1) := FLO_6(COEF(1));
      TOTAL(2) := FLO_6(COEF(2));
      VECTOR_IO.WRITE (D, TOTAL);
   end loop;

   TOTAL(0) := -1.0;
   VECTOR_IO.WRITE (D, (TOTAL(0), 0.0, 0.0, 0.0));

end PROJ1;
```

APPENDIX E

For a single transputer system, there are eight files that are needed to support the operation. Assume all Ada code that will be run on single transputer has the main procedure named PROJ in the file PROJ.ADA.

This appendix provides all code that is necessary to run our Ada program on single transputer, which is the following:

- makefile
- family.inv
- proj.inv
- main.occ
- merger.occ
- projh.occ
- projh2.occ
- main.lnk

```
# File : makefile


# "make help" to print option list
#       Complete development cycle:
#       make family       -- makes Ada family and library directories
#       make              -- compiles, links, configures source
#       make run          -- run bootable code

MODE = s
PROC = 8
OPTS = /$(MODE) /t$(PROC)

# make the executable code
main.b$(PROC)$(MODE): main.c$(PROC)$(MODE)
        iboot main.c$(PROC)$(MODE)
        @ f:\util\bell

main.c$(PROC)$(MODE): proj.o merger.t$(PROC)$(MODE) projh.t$(PROC)$(MODE)
main.t$(PROC)$(MODE)
        ilink /f main.lnk

proj.o: proj.ada
        ada invoke proj.inv,yes

merger.t$(PROC)$(MODE): merger.occ
        occam $(OPTS) merger.occ

projh.t$(PROC)$(MODE): projh2.tax projh.occ
        occam $(OPTS) projh.occ

projh2.tax: projh2.occ
        occam /ta /x projh2.occ

main.t$(PROC)$(MODE): main.occ
        occam $(OPTS) main.occ

#
# misc.
#
help:
        @ echo Make arguments:
        @ echo   make               - make from top level down
```

```
        @ echo   make -n [opt]     - display but don't execute commands
        @ echo   make proj.o       - make Ada object
        @ echo   make help         - display this list
        @ echo   make clean        - delete all files except source
        @ echo   make run          - run bootable program
        @ echo   make check        - check transputer topology
        @ echo   make family       - make Ada family and library directories

clean:
        del *.?8?
        del *.tax
        del *.o
        del proj_lib\adalib.*
        rd proj_lib
        del proj_fam\adafam.*
        rd proj_fam

run:
        iserver /sb main.b8s

check:
        check /r

family:
        ada invoke family.inv,yes


-- File: family.inv

family.new proj_fam,overwrite=yes
lib(family=proj_fam).new proj_lib,overwrite=yes

-- File: proj.inv

compile source=proj.ada,library=proj_lib
default.bind library=proj_lib,level=bind,warning=no
bind proj,object="proj.o",entry_point="proj.program"


#File : main.occ

#OPTION "AGNVW"
#INCLUDE "hostio.inc"
```

```
PROC MAIN.ENTRY (CHAN OF SP FromFiler, ToFiler, []INT FreeMemory,
                 StackMemory)

  #USE "hostio.lib"

  #USE "merger.t8s"
  #USE "projh.t8s"

  [1]CHAN OF ANY Debug:
  [2]CHAN OF SP FromAda, ToAda:
  CHAN OF BOOL StopDebug, StopMultiplexor:
  WHILE TRUE
    SEQ

      PAR

      -- A multiplexor to combine the debug and normal output.
      so.multiplexor (FromFiler, ToFiler, FromAda, ToAda, StopMultiplexor)

      -- A debug channel merger.
      debug.merger (ToAda[0], FromAda[0], Debug, StopDebug)

      -- A process to invoke the Ada programs.
      SEQ
        PAR
          [50000] INT ws:
          proj.harness (FromAda[1], ToAda[1], Debug[0], ws)
          StopDebug ! FALSE
          StopMultiplexor ! FALSE

      so.exit (FromFiler, ToFiler, sps.success)
:
```

```
# File: merger.occ

  #OPTION "AGNVW"
  #INCLUDE "hostio.inc"

  PROC debug.merger (CHAN OF SP FromFiler, ToFiler,
                     []CHAN OF ANY Debug,
                     CHAN OF BOOL Stop)

  #USE "hostio.lib"

  -- A debug channel merger and blocker.

  VAL max.debug IS 20:
  VAL number.of.debug IS SIZE Debug:

  INT line.index:
  [256]BYTE line.buffer:
  BYTE value, r:
  BOOL running, reset, s:
  [max.debug]BOOL mask:
  VAL BYTE line.feed IS 10 (BYTE):
  SEQ
    SEQ i = 0 FOR number.of.debug
      mask[i] := TRUE
    running := TRUE
    reset := FALSE
    line.index := 0
    WHILE running
      PRI ALT
        ALT i = 0 FOR number.of.debug
          mask[i] & Debug[i] ? value
            SEQ
              IF
                value = line.feed
                  SEQ
                    -- Send the complete line.
                    so.puts (FromFiler, ToFiler, spid.stdout,
                      [line.buffer FROM 0 FOR line.index], r)
                    line.index := 0
                    mask [i] := FALSE
                    reset := TRUE
```

```
            TRUE
              SEQ
                -- Add character to line.
                line.buffer[line.index] := value
                line.index := line.index + 1
        reset & SKIP
          SEQ
            reset := FALSE
            SEQ i = 0 FOR number.of.debug
              mask[i] := TRUE
        Stop ? s
          running := FALSE
:


-- File: main.lnk
-- Purpose: File list for ilink
   main.t8s
   merger.t8s
   hostio.lib
   occam8s.lib
   ( projh.t8s proj.o adarts8.lib hostio.lib occam8s.lib )


# File: projh2.occ

   #OPTION "AEV"

   PROC proj.program ([]INT ws1, in, out, ws2)
  [1000]INT d:
  SEQ
    SKIP
:
```

```
# File: projh.occ

  #OPTION "AGNVW"
  #INCLUDE "hostio.inc"

  PROC proj.harness (CHAN OF SP  FromAda, ToAda,
                     CHAN OF ANY Debug,
                     []INT FreeMemory)

#IMPORT "projh2.tax"

[1]INT dummy.ws:
ws IS FreeMemory:
[2]INT in.program, out.program:
SEQ
  -- Set up vector of pointers to channels.
  in.program[0] := MOSTNEG INT    -- not used
  LOAD.INPUT.CHANNEL (in.program[1], ToAda)
  LOAD.OUTPUT.CHANNEL (out.program[0], Debug)
  LOAD.OUTPUT.CHANNEL (out.program[1], FromAda)
  -- Invoke the Ada program.
  -- Assumes the entry point name has been changed to "proj.program".
  proj.program (ws, in.program, out.program, dummy.ws)
:
```

APPENDIX F

In multiple transputer systems, there is a need for the Occam Harness to support the Ada program. Each Ada program run on individual transputer needs its own mini-harness. This appendix provides all support code for the network of five transputers. For general purpose of this support code, assume that the Ada program run on transputer T_0 has the main procedure named PROJ0 and is in the file PRPJ0.ADA, the Ada program runs on transputer T_1 has the main procedure named PROJ1 and is in the file PROJ1.ADA, and so on. All the support code is the following:

- makefile
- family.inv
- proj?.inv 5 files
- mainh.occ
- merger.occ
- proj?h.occ 5 files
- proj?h2.occ 5 files
- main.pgm

```
# File: makefile


#   "make help" to print option list
#
#   Complete development cycle:
#     make family       -- makes Ada family and library directories
#     make              -- compiles, links, configures source
#     make run          -- run bootable code

MODE = s
PROC = 8
OPTS = /$(MODE) /t$(PROC)

# make the executable code
main.btl:  mainh.c$(PROC)$(MODE)  proj1h.c$(PROC)$(MODE)
proj2h.c$(PROC)$(MODE)  proj3h.c$(PROC)$(MODE)  proj4h.c$(PROC)$(MODE)
main.pgm
   @ echo EXPECT 1 WARNING... PRAY FOR !!!.. SUCCESSFUL
   iconf /s main.pgm
   @ f:\util\bell

mainh.c$(PROC)$(MODE):  proj0.o  proj0h.t$(PROC)$(MODE)
merger.t$(PROC)$(MODE) mainh.t$(PROC)$(MODE)
    ilink  mainh.t$(PROC)$(MODE)  proj0.o  proj0h.t$(PROC)$(MODE)
merger.t$(PROC)$(MODE) adarts8.lib hostio.lib occam8s.lib

proj0.o: common.ada proj0.ada
   ada invoke proj0.inv,yes

proj0h.t$(PROC)$(MODE): proj0h2.tax proj0h.occ
   occam $(OPTS) proj0h.occ

proj0h2.tax: proj0h2.occ
   occam /ta /x proj0h2.occ

merger.t$(PROC)$(MODE): merger.occ
   occam $(OPTS) merger.occ

mainh.t$(PROC)$(MODE): mainh.occ
   occam $(OPTS) mainh.occ
proj1h.c$(PROC)$(MODE): proj1.o proj1h.t$(PROC)$(MODE)
   ilink proj1h.t$(PROC)$(MODE) proj1.o adarts8.lib hostio.lib occam8s.lib
```

```
proj1.o: common.ada proj1.ada
    ada invoke proj1.inv,yes
proj1h.t$(PROC)$(MODE): proj1h2.tax proj1h.occ
    occam $(OPTS) proj1h.occ
proj1h2.tax: proj1h2.occ
    occam /ta /x proj1h2.occ

proj2h.c$(PROC)$(MODE): proj2.o proj2h.t$(PROC)$(MODE)
    ilink proj2h.t$(PROC)$(MODE) proj2.o adarts8.lib hostio.lib occam8s.lib

proj2.o: common.ada proj2.ada
    ada invoke proj2.inv,yes
proj2h.t$(PROC)$(MODE): proj2h2.tax proj2h.occ
    occam $(OPTS) proj2h.occ
proj2h2.tax: proj2h2.occ
    occam /ta /x proj2h2.occ

proj3h.c$(PROC)$(MODE): proj3.o proj3h.t$(PROC)$(MODE)
    ilink proj3h.t$(PROC)$(MODE) proj3.o adarts8.lib hostio.lib occam8s.lib

proj3.o: common.ada proj3.ada
    ada invoke proj3.inv,yes
proj3h.t$(PROC)$(MODE): proj3h2.tax proj3h.occ
    occam $(OPTS) proj3h.occ
proj3h2.tax: proj3h2.occ
    occam /ta /x proj3h2.occ

proj4h.c$(PROC)$(MODE): proj4.o proj4h.t$(PROC)$(MODE)
    ilink proj4h.t$(PROC)$(MODE) proj4.o adarts8.lib hostio.lib occam8s.lib

proj4.o: common.ada proj4.ada
    ada invoke proj4.inv,yes
proj4h.t$(PROC)$(MODE): proj4h2.tax proj4h.occ
    occam $(OPTS) proj4h.occ
proj4h2.tax: proj4h2.occ
    occam /ta /x proj4h2.occ

# misc.

run:    iserver /sb main.btl

family:  ada invoke family.inv,yes
```

```
-- File: proj0.inv

   default.compile library=test_lib
   compile common.ada
   compile proj0.ada
   default.bind library=test_lib,level=bind,warning=no
   bind proj0,object="proj0.o",entry_point="proj0.program"


-- File: proj1.inv

   default.compile library=test_lib
   compile common.ada
   compile proj1.ada
   default.bind library=test_lib,level=bind,warning=no
   bind proj1,object="proj1.o",entry_point="proj1.program"


-- File: proj2.inv

   default.compile library=test_lib
   compile common.ada
   compile proj2.ada
   default.bind library=test_lib,level=bind,warning=no
   bind proj2,object="proj2.o",entry_point="proj2.program"


-- File: proj3.inv

   default.compile library=test_lib
   compile common.ada
   compile proj3.ada
   default.bind library=test_lib,level=bind,warning=no
   bind proj3,object="proj3.o",entry_point="proj3.program"


-- File: proj4.inv

   default.compile library=test_lib
   compile common.ada
   compile proj4.ada
   default.bind library=test_lib,level=bind,warning=no
   bind proj4,object="proj4.o",entry_point="proj4.program"
```

```
-- File: family.inv

   family.new test_fam,overwrite=yes
   lib(family=test_fam).new test_lib,overwrite=yes


-- File: mainh.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC main.harness (CHAN OF SP  FromFiler, ToFiler,
                   CHAN OF INT AdaChannel,
                   []INT FreeMemory)

  #USE "hostio.lib"

  #USE "proj0h.t8s"
  #USE "merger.t8s"

  [1]CHAN OF ANY Debug:
  [2]CHAN OF SP FromAda, ToAda:
  CHAN OF BOOL StopDebug, StopMultiplexor:
  SEQ

    PAR

      -- A multiplexor to combine the debug and normal output.
      so.multiplexor (FromFiler, ToFiler, FromAda, ToAda, StopMultiplexor)

      -- A debug channel merger.
      debug.merger (ToAda[0], FromAda[0], Debug, StopDebug)

      -- A process to invoke the sieve program.
      ws IS FreeMemory:
      SEQ
        proj0.harness (FromAda[1], ToAda[1], Debug[0], AdaChannel, ws)
        StopDebug ! FALSE
        StopMultiplexor ! FALSE

    so.exit (FromFiler, ToFiler, sps.success)
:
```

```
-- File: merger.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC debug.merger (CHAN OF SP FromFiler, ToFiler,
                   []CHAN OF ANY Debug,
                   CHAN OF BOOL Stop)

  #USE "hostio.lib"

  -- A debug channel merger and blocker.

  VAL max.debug IS 20:
  VAL number.of.debug IS SIZE Debug:

  INT line.index:
  [256]BYTE line.buffer:
  BYTE value, r:
  BOOL running, reset, s:
  [max.debug]BOOL mask:
  VAL BYTE line.feed IS 10 (BYTE):
  SEQ
    SEQ i = 0 FOR number.of.debug
      mask[i] := TRUE
    running := TRUE
    reset := FALSE
    line.index := 0
    WHILE running
      PRI ALT
        ALT i = 0 FOR number.of.debug
          mask[i] & Debug[i] ? value
            SEQ
              IF
                value = line.feed
                  SEQ
                    -- Send the complete line.
                    so.puts (FromFiler, ToFiler, spid.stdout,
                      [line.buffer FROM 0 FOR line.index], r)
                    line.index := 0
                    mask [i] := FALSE
                    reset := TRUE
```

```
        TRUE
          SEQ
            -- Add character to line.
            line.buffer[line.index] := value
            line.index := line.index + 1
reset & SKIP
  SEQ
    reset := FALSE
    SEQ i = 0 FOR number.of.debug
      mask[i] := TRUE
Stop ? s
  running := FALSE
:
```

```
-- File: proj0h.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC proj0.harness (CHAN OF SP  FromAda, ToAda,
                    CHAN OF ANY Debug,
                    CHAN OF INT AdaChannel,
                    []INT FreeMemory)


  #IMPORT "proj0h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  [3]INT in.program:
  [2]INT out.program:
  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT    -- not used

    LOAD.INPUT.CHANNEL (in.program[1], ToAda)
    LOAD.INPUT.CHANNEL (in.program[2], AdaChannel)
    LOAD.OUTPUT.CHANNEL (out.program[0], Debug)
    LOAD.OUTPUT.CHANNEL (out.program[1], FromAda)
    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to "proj0.program".
    proj0.program (ws1, in.program, out.program, dummy.ws)

:


-- File: proj0h2.occ


#OPTION "AEV"
PROC proj0.program ([]INT ws1, in, out, ws2)
   [1000]INT d:
   SEQ
      SKIP
:
```

```
-- File: proj1h.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC proj1.harness (CHAN OF INT AdaChannel,
                    CHAN OF INT AdaChan,
                    []INT FreeMemory)

  #IMPORT "proj1h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  ws2 IS FreeMemory:
  [3]INT in.program:
  [3]INT out.program:

  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT    -- not used

    LOAD.INPUT.CHANNEL (in.program[2], AdaChan)
    LOAD.OUTPUT.CHANNEL (out.program[2], AdaChannel)

    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to "proj1.program".
    proj1.program (ws1, in.program, out.program, ws2)
:


-- File: proj1h2.occ


#OPTION "AEV"
PROC proj1.program ([]INT ws1, in, out, ws2)
   [1000]INT d:
   SEQ
      SKIP
:
```

```
-- File: proj2h.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC proj2.harness (CHAN OF INT AdaChan,
                    CHAN OF INT Achan,
                    []INT FreeMemory)

  #IMPORT "proj2h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  ws2 IS FreeMemory:
  [3]INT in.program:
  [3]INT out.program:

  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT    -- not used

    LOAD.INPUT.CHANNEL (in.program[2], Achan)
    LOAD.OUTPUT.CHANNEL (out.program[2], AdaChan)

    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to "proj2.program".
    proj2.program (ws1, in.program, out.program, ws2)
:


-- File: proj2h2.occ


#OPTION "AEV"
PROC proj2.program ([]INT ws1, in, out, ws2)
   [1000]INT d:
   SEQ
      SKIP
:
```

```
-- File: proj3h.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC proj3.harness (CHAN OF INT Achan,
                    CHAN OF INT Chan,
                    []INT FreeMemory)

  #IMPORT "proj3h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  ws2 IS FreeMemory:
  [3]INT in.program:
  [3]INT out.program:

  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT    -- not used

    LOAD.INPUT.CHANNEL (in.program[2], Chan)
    LOAD.OUTPUT.CHANNEL (out.program[2], Achan)

    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to "proj3.program".
    proj3.program (ws1, in.program, out.program, ws2)
:


-- File: proj3h2.occ


#OPTION "AEV"
PROC proj3.program ([]INT ws1, in, out, ws2)
   [1000]INT d:
   SEQ
      SKIP
:
```

```
-- File: proj4h.occ


#OPTION "AGNVW"
#INCLUDE "hostio.inc"

PROC proj4.harness (CHAN OF INT Chan,
                    []INT FreeMemory)

  #IMPORT "proj4h2.tax"

  [1]INT dummy.ws:
  ws1 IS FreeMemory:
  [2]INT in.program:
  [3]INT out.program:
  SEQ
    -- Set up vector of pointers to channels.
    in.program[0] := MOSTNEG INT     -- not used
    in.program[1] := MOSTNEG INT     -- standard i/o not used
    out.program[0] := MOSTNEG INT    -- standard i/o not used
    out.program[1] := MOSTNEG INT    -- standard i/o not used

    LOAD.OUTPUT.CHANNEL (out.program[2], Chan)
    -- Invoke the Ada program.
    -- Assumes the entry point name has been changed to "proj3.program".
    proj4.program (ws1, in.program, out.program, dummy.ws)
:


-- File: proj4h2.occ


#OPTION "AEV"
PROC proj4.program ([]INT ws1, in, out, ws2)
   [1000]INT d:
   SEQ
      SKIP
:
```

```
# File: main.pgm


#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"

#USE "mainh.c8s"
#USE "proj1h.c8s"
#USE "proj2h.c8s"
#USE "proj3h.c8s"
#USE "proj4h.c8s"

CHAN OF INT AdaChannel:
CHAN OF INT AdaChan:
CHAN OF INT Achan:
CHAN OF INT Chan:
CHAN OF SP FromFiler, ToFiler:

PLACED PAR

  PROCESSOR 0 T8
    PLACE FromFiler   AT link0.in:
    PLACE ToFiler     AT link0.out:
    PLACE AdaChannel  AT link2.in:

    [100000] INT ws1:
    main.harness (FromFiler, ToFiler, AdaChannel, ws1)

  PROCESSOR 1 T8

    PLACE AdaChannel  AT link0.out:
    PLACE AdaChan     AT link2.in:

    [100000] INT ws2:
    proj1.harness (AdaChannel, AdaChan, ws2)

  PROCESSOR 2 T8

    PLACE AdaChan AT link3.out:
    PLACE Achan   AT link2.in:

    [100000] INT ws3:
    proj2.harness (AdaChan, Achan, ws3)
```

```
PROCESSOR 3 T8

  PLACE Achan  AT link3.out:
  PLACE Chan AT link2.in:

  [100000] INT ws4:
  proj3.harness (Achan, Chan, ws4)

PROCESSOR 4 T8

  PLACE Chan AT link3.out:

  [100000] INT ws5:
  proj4.harness (Chan, ws5)
```

APPENDIX G

This appendix provides the Ada code for a five transputers network. Each file is the Ada program that runs on transputer T_0, T_1, T_2, T_3, T_4, and T_5 individually. In this appendix also provides the package COMMON in file COMMON.ADA which is the declaration of common data types used for channel communication.

```
-- File: common.ada

with CHANNELS;

package COMMON is

   -- Declarations of common data types, especially those used for channel
   -- communication.

   type FLO_6 is digits 6
         range -(2.0 - 2.0**(-23))*2.0**127 .. (2.0 - 2.0**(-23))*2.0**127;

   type VECTOR is array (0..2) of FLO_6;

   -- Instantiations of the generic channel i/o package.

   package VECTOR_IO is new CHANNELS.CHANNEL_IO (VECTOR);

end COMMON;
```

```
-- File: proj0.ada


with TEXT_IO, COMMON, CHANNELS;
use COMMON;

procedure PROJ0 is


   package FLO_IO is new TEXT_IO.FLOAT_IO(FLO_6);
   use FLO_IO;

   RESULT : VECTOR;
   RES    : VECTOR;
   TOTAL  : VECTOR;
   TOL    : VECTOR;
   SUM    : VECTOR;

   D : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   -- The channel used for communication with the program producing the stream.
   -- Input channels 0 and 1 are reserved for use by the run-time system.

begin

   for J in 1..5 loop
      SUM := (500.00,500.00,500.00);
      TEXT_IO.PUT ("DATA FROM TRANSPUTER # 0 " );
      FLO_IO.PUT (SUM(0), FORE => 4, AFT => 2, EXP => 0);
      FLO_IO.PUT (SUM(1), FORE => 4, AFT => 2, EXP => 0);
      FLO_IO.PUT (SUM(2), FORE => 4, AFT => 2, EXP => 0);
      TEXT_IO.NEW_LINE;
   end loop;

   loop
      VECTOR_IO.READ (D, TOTAL);
      exit when TOTAL(0) < 0.0;
      TEXT_IO.PUT ("DATA FROM TRANSPUTER # 1 " );
      FLO_IO.PUT (TOTAL(0), FORE => 4, AFT => 2, EXP =>0);
      FLO_IO.PUT (TOTAL(1), FORE => 4, AFT => 2, EXP =>0);
      FLO_IO.PUT (TOTAL(2), FORE => 4, AFT => 2, EXP =>0);
      TEXT_IO.NEW_LINE;
   end loop;
```

```
    loop
       VECTOR_IO.READ (D, TOL);
       exit when TOL(0) < 0.0;
       TEXT_IO.PUT ("DATA FROM TRANSPUTER # 2 " );
       FLO_IO.PUT (TOL(0), FORE => 4, AFT => 2, EXP =>0);
       FLO_IO.PUT (TOL(1), FORE => 4, AFT => 2, EXP =>0);
       FLO_IO.PUT (TOL(2), FORE => 4, AFT => 2, EXP =>0);
       TEXT_IO.NEW_LINE;
    end loop;

    loop
       VECTOR_IO.READ (D, RES);
       exit when RES(0) < 0.0;
       TEXT_IO.PUT ("DATA FROM TRANSPUTER # 3 " );
       FLO_IO.PUT (RES(0), FORE => 4, AFT => 2, EXP => 0);
       FLO_IO.PUT (RES(1), FORE => 4, AFT => 2, EXP => 0);
       FLO_IO.PUT (RES(2), FORE => 4, AFT => 2, EXP => 0);
       TEXT_IO.NEW_LINE;
    end loop;

    loop
       VECTOR_IO.READ (D, RESULT);
       exit when RESULT(0) < 0.0;
       TEXT_IO.PUT ("DATA FROM TRANSPUTER # 4 " );
       FLO_IO.PUT (RESULT(0), FORE => 4, AFT => 2, EXP => 0);
       FLO_IO.PUT (RESULT(1), FORE => 4, AFT => 2, EXP => 0);
       FLO_IO.PUT (RESULT(2), FORE => 4, AFT => 2, EXP => 0);
       TEXT_IO.NEW_LINE;
    end loop;

end PROJ0;
```

```
-- File: proj1.ada


with TEXT_IO, COMMON, CHANNELS;
use COMMON;

procedure PROJ1 is

   RESULT : VECTOR;
   RES    : VECTOR;
   TOTAL  : VECTOR;
   TOL    : VECTOR;

   C : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   D : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel used for communication with the program producing the stream.

begin
   for J in 1..5 loop
      TOTAL := (400.00,400.00,400.00);
      VECTOR_IO.WRITE (D, TOTAL);
   end loop;
   TOTAL(0) := -1.00;
   VECTOR_IO.WRITE (D, (TOTAL(0),0.00,0.00));

   loop
      VECTOR_IO.READ (C, TOL);
      VECTOR_IO.WRITE (D, TOL);
      exit when TOL(0) < 0.0;
   end loop;

   loop
      VECTOR_IO.READ (C, RES);
      VECTOR_IO.WRITE (D, RES);
      exit when RES(0) < 0.0;
   end loop;

   loop
      VECTOR_IO.READ (C, RESULT);
      VECTOR_IO.WRITE (D, RESULT);
      exit when RESULT(0) < 0.0;
   end loop;
end PROJ1;
```

```
-- File: proj2.ada


with TEXT_IO;
with COMMON;
with CHANNELS;

procedure PROJ2 is

   use COMMON;

   RESULT : VECTOR;
   RES : VECTOR;
   TOL : VECTOR;

   B : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   C : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel for communication with the other Ada program.
   -- Output channels 0 and 1 are reserved for use by the run-time system.

begin

   for I in 1 .. 5 loop
      TOL := (300.00,300.00,300.00);
      VECTOR_IO.WRITE (C, TOL);
   end loop;

   TOL(0) := -1.00;
   VECTOR_IO.WRITE (C, (TOL(0),0.0,0.0));

   loop
      VECTOR_IO.READ (B, RESULT);
      VECTOR_IO.WRITE (C, RESULT);
      exit when RESULT(0) < 0.0;
   end loop;

   loop
      VECTOR_IO.READ (B, RES);
      VECTOR_IO.WRITE (C, RES);
      exit when RES(0) < 0.0;
   end loop;

end PROJ2;
```

```
-- File: proj3.ada


with TEXT_IO;
with COMMON;
with CHANNELS;

procedure PROJ3 is

   use COMMON;

   RESULT : VECTOR;
   RES : VECTOR;

   A : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   B : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel for communication with the other Ada program.
   -- Output channels 0 and 1 are reserved for use by the run-time system.

begin

   for I in 1 .. 5 loop
      RES := (200.00,200.00,200.00);
      VECTOR_IO.WRITE (B, RES);
   end loop;

   RES(0) := -1.00;
   VECTOR_IO.WRITE (B, (RES(0),0.0,0.0));

   loop
      VECTOR_IO.READ (A, RESULT);
      VECTOR_IO.WRITE (B, RESULT);
      exit when RESULT(0) < 0.0;
   end loop;

end PROJ3;
```

```
-- File: proj4.ada


with TEXT_IO;
with COMMON;
with CHANNELS;

procedure PROJ4 is

   use COMMON;

   RESULT : VECTOR;

   A : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel for communication with the other Ada program.
   -- Output channels 0 and 1 are reserved for use by the run-time system.

begin

   for I in 1 .. 5 loop
      RESULT := (100.00,100.00,100.00);
      VECTOR_IO.WRITE (A, RESULT);
   end loop;

   RESULT(0) := -1.00;
   VECTOR_IO.WRITE (A, (RESULT(0),0.0,0.0));

end PROJ4;
```

```
Booting root transputer...ok
DATA FROM TRANSPUTER # 0  500.00 500.00 500.00
DATA FROM TRANSPUTER # 0  500.00 500.00 500.00
DATA FROM TRANSPUTER # 0  500.00 500.00 500.00
DATA FROM TRANSPUTER # 0  500.00 500.00 500.00
DATA FROM TRANSPUTER # 0  500.00 500.00 500.00
DATA FROM TRANSPUTER # 1  400.00 400.00 400.00
DATA FROM TRANSPUTER # 1  400.00 400.00 400.00
DATA FROM TRANSPUTER # 1  400.00 400.00 400.00
DATA FROM TRANSPUTER # 1  400.00 400.00 400.00
DATA FROM TRANSPUTER # 1  400.00 400.00 400.00
DATA FROM TRANSPUTER # 2  300.00 300.00 300.00
DATA FROM TRANSPUTER # 2  300.00 300.00 300.00
DATA FROM TRANSPUTER # 2  300.00 300.00 300.00
DATA FROM TRANSPUTER # 2  300.00 300.00 300.00
DATA FROM TRANSPUTER # 2  300.00 300.00 300.00
DATA FROM TRANSPUTER # 3  200.00 200.00 200.00
DATA FROM TRANSPUTER # 3  200.00 200.00 200.00
DATA FROM TRANSPUTER # 3  200.00 200.00 200.00
DATA FROM TRANSPUTER # 3  200.00 200.00 200.00
DATA FROM TRANSPUTER # 3  200.00 200.00 200.00
DATA FROM TRANSPUTER # 4  100.00 100.00 100.00
DATA FROM TRANSPUTER # 4  100.00 100.00 100.00
DATA FROM TRANSPUTER # 4  100.00 100.00 100.00
DATA FROM TRANSPUTER # 4  100.00 100.00 100.00
DATA FROM TRANSPUTER # 4  100.00 100.00 100.00
MAKE Program Maintenance Utility, Logical Systems Version 89.1
iserver /sb main.btl
```

APPENDIX H

This appendix provides the code of Small Tactical System in a three transputers network. Since this is also the multiple transputer network, the support code in APPENDIX F can be used. There is only a few change in file "makefile" and "main.pgm" which is shown in this appendix.

```
-- File: common.ada


with CHANNELS;

package COMMON is

   -- Declarations of common data types, especially those used for channel
   -- communication.

    type FLO_6 is digits 6
       range -(2.0 - 2.0**(-23))*2.0**127 .. (2.0 - 2.0**(-23))*2.0**127;

    type COMPONENT is (X,Y,Z);
    type VECTOR is array (COMPONENT) of FLO_6;
    type ARY_7 is array (0..6) of VECTOR;
    type VEC_3 is array (0..2) of FLO_6;

   -- Instantiations of the generic channel i/o package.

   package VECTOR_IO is new CHANNELS.CHANNEL_IO (VEC_3);
   package ARRAY_IO is new CHANNELS.CHANNEL_IO (ARY_7);

end COMMON;
```

```
-- File: proj0.ada


with TEXT_IO, COMMON, CHANNELS;

procedure PROJ0 is

   use COMMON;

   package FLO_IO is new TEXT_IO.FLOAT_IO(FLO_6);
   use FLO_IO;

   RESULT : VEC_3;
   TOTAL  : VEC_3;
   SUM    : VEC_3;

   D : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   -- The channel used for communication with the program producing the stream.
   -- Input channels 0 and 1 are reserved for use by the run-time system.

begin

  for J in 1..5 loop
      SUM := (500.00,500.00,500.00);
      TEXT_IO.PUT ("DATA FROM TRANSPUTER # 0 " );
      FLO_IO.PUT (SUM(0), FORE => 6, AFT => 4, EXP => 0);
      FLO_IO.PUT (SUM(1), FORE => 6, AFT => 4, EXP => 0);
      FLO_IO.PUT (SUM(2), FORE => 6, AFT => 4, EXP => 0);
      TEXT_IO.NEW_LINE;
  end loop;

  loop
    VECTOR_IO.READ (D, TOTAL);
    exit when TOTAL(0) < 0.0;
    TEXT_IO.PUT ("DATA X FROM TRANSPUTER # 1 " );
    FLO_IO.PUT (TOTAL(0), FORE => 6, AFT => 4, EXP => 0);
    FLO_IO.PUT (TOTAL(1), FORE => 6, AFT => 4, EXP => 0);
    FLO_IO.PUT (TOTAL(2), FORE => 6, AFT => 4, EXP => 0);
    TEXT_IO.NEW_LINE;

    VECTOR_IO.READ (D, TOTAL);
    TEXT_IO.PUT ("DATA Y FROM TRANSPUTER # 1 " );
    FLO_IO.PUT (TOTAL(0), FORE => 6, AFT => 4, EXP => 0);
```

```
    FLO_IO.PUT (TOTAL(1), FORE => 6, AFT => 4, EXP => 0);
    FLO_IO.PUT (TOTAL(2), FORE => 6, AFT => 4, EXP => 0);
    TEXT_IO.NEW_LINE;

    VECTOR_IO.READ (D, TOTAL);
    TEXT_IO.PUT ("DATA Z FROM TRANSPUTER # 1 " );
    FLO_IO.PUT (TOTAL(0), FORE => 6, AFT => 4, EXP => 0);
    FLO_IO.PUT (TOTAL(1), FORE => 6, AFT => 4, EXP => 0);
    FLO_IO.PUT (TOTAL(2), FORE => 6, AFT => 4, EXP => 0);
    TEXT_IO.NEW_LINE;

  end loop;

end PROJ0;
```

```
-- File: proj1.ada


with TEXT_IO, COMMON, CHANNELS;
use  TEXT_IO, COMMON;

procedure PROJ1 is

   B : ARY_7;
   TOTAL  : VEC_3;

   C : CHANNELS.CHANNEL_REF := CHANNELS.IN_PARAMETERS (2);
   D : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
   -- The channel used for communication with the program producing the stream.
   -- Input channels 0 and 1 are reserved for use by the run-time system.

    -- type identification
    type COMPONENTS is (X0,X1,X2,X3,X4,X5,X6);
    type VECTORS is array (COMPONENTS) of FLO_6;
    type POSITIONS is array (0..3) of FLO_6;
    type VEC is array (0..2) of FLO_6;

    -- orthogonal constant
    CONST0 : VECTORS := (1.0,1.0,1.0,1.0,1.0,1.0,1.0);
    CONST1 : VECTORS := (1.0,0.666,0.333,0.0,-0.333,-0.666,-1.0);
    CONST2 : VECTORS := (1.0,0.0,-0.6,-0.8,-0.6,0.0,1.0);
    CONST3 : VECTORS := (1.0,-1.0,-1.0,0.0,1.0,1.0,-1.0);
    T,U,V  : VECTORS   := (0.0,0.0,0.0,0.0,0.0,0.0,0.0);
    POS_X,POS_Y,POS_Z  : POSITIONS := (0.0,0.0,0.0,0.0);
    COEF : VEC;

    function VECTOR_SUM (X : VECTORS) return FLO_6 is
       SUM : FLO_6 := 0.0;
    begin
       for I in X'RANGE loop
          SUM := SUM + X(I);
       end loop;
       return SUM;
    end VECTOR_SUM;
```

```
   function VECTOR_SUM_SQUARE (X : VECTORS) return FLO_6 is
    SUM_SQUARE : FLO_6 := 0.0;
begin
    for I in X'RANGE loop
        SUM_SQUARE := SUM_SQUARE + (X(I)**2);
    end loop;
    return SUM_SQUARE;
end VECTOR_SUM_SQUARE;

function ORTHOGONAL (POS_IN : VECTORS) return POSITIONS is
    TEMP0,TEMP1,TEMP2,TEMP3 : VECTORS;
    POS_OUT0,POS_OUT1,POS_OUT2,POS_OUT3 : FLO_6;
    ORTHO : POSITIONS;
begin
    for I in COMPONENTS loop
        TEMP0(I) := POS_IN(I) * CONST0(I);
        TEMP1(I) := POS_IN(I) * CONST1(I);
        TEMP2(I) := POS_IN(I) * CONST2(I);
        TEMP3(I) := POS_IN(I) * CONST3(I);
    end loop;
    POS_OUT0 := VECTOR_SUM(TEMP0)/VECTOR_SUM_SQUARE(CONST0);
    POS_OUT1 := VECTOR_SUM(TEMP1)/VECTOR_SUM_SQUARE(CONST1);
    POS_OUT2 := VECTOR_SUM(TEMP2)/VECTOR_SUM_SQUARE(CONST2);
    POS_OUT3 := VECTOR_SUM(TEMP3)/VECTOR_SUM_SQUARE(CONST3);
    ORTHO := (POS_OUT0,POS_OUT1,POS_OUT2,POS_OUT3);
    return ORTHO;
end ORTHOGONAL;

function COEFF_CAL (A : in POSITIONS) return VEC is
    a0,a1,a2 : FLO_6;   V : VEC;
begin
    a0 := A(0)+A(1)+A(2)+A(3);
    a1 := (-A(1)*(0.3333))+(A(2)*(-1.2))+(-A(3)*(0.6667));
    a2 := (A(2)*(0.2000))+(A(3)*(1.500));
    V := (a0,a1,a2);
    return V;
end COEFF_CAL;
```

```
begin
   loop
      ARRAY_IO.READ (C, B);
      exit when B(0)(X) < 0.0;

      T(X0) := B(0)(X);   U(X0) := B(0)(Y);   V(X0) := B(0)(Z);
      T(X1) := B(1)(X);   U(X1) := B(1)(Y);   V(X1) := B(1)(Z);
      T(X2) := B(2)(X);   U(X2) := B(2)(Y);   V(X2) := B(2)(Z);
      T(X3) := B(3)(X);   U(X3) := B(3)(Y);   V(X3) := B(3)(Z);
      T(X4) := B(4)(X);   U(X4) := B(4)(Y);   V(X4) := B(4)(Z);
      T(X5) := B(5)(X);   U(X5) := B(5)(Y);   V(X5) := B(5)(Z);
      T(X6) := B(6)(X);   U(X6) := B(6)(Y);   V(X6) := B(6)(Z);

      POS_X := ORTHOGONAL(T);
      COEF := COEFF_CAL(POS_X);
      TOTAL(0) := FLO_6(COEF(0));
      TOTAL(1) := FLO_6(COEF(1));
      TOTAL(2) := FLO_6(COEF(2));
      VECTOR_IO.WRITE (D, TOTAL);

      POS_Y := ORTHOGONAL(U);
      COEF := COEFF_CAL(POS_Y);
      TOTAL(0) := FLO_6(COEF(0));
      TOTAL(1) := FLO_6(COEF(1));
      TOTAL(2) := FLO_6(COEF(2));
      VECTOR_IO.WRITE (D, TOTAL);

      POS_Z := ORTHOGONAL(V);
      COEF := COEFF_CAL(POS_Z);
      TOTAL(0) := FLO_6(COEF(0));
      TOTAL(1) := FLO_6(COEF(1));
      TOTAL(2) := FLO_6(COEF(2));
      VECTOR_IO.WRITE (D, TOTAL);

   end loop;

   TOTAL(0) := -1.0;
   VECTOR_IO.WRITE (D, (TOTAL(0), 0.0, 0.0));

end PROJ1;
```

```
-- File: proj2.ada

with TEXT_IO, CALENDAR, GENERIC_ELEMENTARY_FUNCTIONS;
with COMMON, CHANNELS;
use  TEXT_IO, CALENDAR, COMMON;


procedure PROJ2 is

   package MATH_FUNCT is new GENERIC_ELEMENTARY_FUNCTIONS(FLO_6);
   use MATH_FUNCT;

  C : CHANNELS.CHANNEL_REF := CHANNELS.OUT_PARAMETERS (2);
  -- The channel for communication with the other Ada program.
  -- Output channels 0 and 1 are reserved for use by the run-time system.

   -- Type identifications
   type VELOCITIES is array (0..4) of VECTOR;

   DELTA_TIME : constant FLO_6 := 0.25;  -- delta time t = 1/4 seconds
   NO_TARGET : BOOLEAN := FALSE;
   CURRENT   : constant INTEGER := 4;
   LINT_SEC  : INTEGER;
   INTERVAL  : DAY_DURATION := 1.0;
   DISP_TIME : DAY_DURATION := 0.0;
   POSITION  : VECTOR := (27000.0,22000.0,5000.0);
   P,P0,P1,P2,P3,P4,P5,P6 : VECTOR;
   INT_VEL   : VECTOR := (230.0,180.0,25.0);
   VELOCITY  : VELOCITIES;
   B  : ARY_7;

   function "+"(LEFT, RIGHT : in VECTOR) return VECTOR is
   ------------------------------------------------------------------
   -- This function is written to handle VECTOR addition
   ------------------------------------------------------------------
      TEMP : VECTOR;
   begin
      TEMP(X) := LEFT(X) + RIGHT(X);
      TEMP(Y) := LEFT(Y) + RIGHT(Y);
      TEMP(Z) := LEFT(Z) + RIGHT(Z);
      return TEMP;
   end "+";
```

```
function "-"(LEFT, RIGHT : in VECTOR) return VECTOR is
-------------------------------------------------------------------------
-- This function is written to handle VECTOR subtraction
-------------------------------------------------------------------------
   TEMP : VECTOR;
begin
   TEMP(X) := LEFT(X) - RIGHT(X);
   TEMP(Y) := LEFT(Y) - RIGHT(Y);
   TEMP(Z) := LEFT(Z) - RIGHT(Z);
   return TEMP;
end "-";

function SIMPSON(XPOS: in VECTOR; VEL: in VELOCITIES) return VECTOR is
-------------------------------------------------------------------------------------------------
-- This function performs numeric integration using SIMPSONS rule and
-- return a position vector giving a set of sample VELOCITIES and the
-- DELTA_TIME between those velocities.
-------------------------------------------------------------------------------------------------
    T : VECTOR;
    I : COMPONENT;
    J : INTEGER;
begin
    for I in COMPONENT loop
        T(I) := (VEL(VELOCITIES'FIRST) (I) + VEL(VELOCITIES'LAST) (I));
        for J in VELOCITIES'FIRST+1..VELOCITIES'LAST-1 loop
            if (J MOD 2) = 1 then
                T(I) := T(I) + 4.0*VEL(J) (I);
            else
                T(I) := T(I) + 2.0*VEL(J) (I);
            end if;
        end loop;
        T(I) :=  DELTA_TIME * T(I) / 3.0;
    end loop;
    return T;
end SIMPSON;

   package ATOD is
-------------------------------------------------------------------------------------------------
-- This package is used to maintain the velocity values for the previous second at 1/4
-- second intervals. It has one function which returns an array of five vectors.
-- The task is written to handle concurrent processing of ACCELEROMETER.
-------------------------------------------------------------------------------------------------
```

```
    procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR);
    procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES);

    task ACCELEROMETER is
       entry START;
    end ACCELEROMETER;

end ATOD;

package body ATOD is

    VEL : VELOCITIES := (others => (others => 0.0));

    procedure INITIALIZE_VELOCITY(FIRST_VEL : in VECTOR) is
       I : INTEGER;
    begin
       for I in VELOCITIES'RANGE loop
          VEL(I) := FIRST_VEL;
       end loop;
    end INITIALIZE_VELOCITY;

    procedure GET_VELOCITIES(NEW_VEL : out VELOCITIES) is
    begin
       NEW_VEL := VEL;
    end GET_VELOCITIES;

    task body ACCELEROMETER is
       use CALENDAR;
       INTERVAL  : constant DURATION := 0.25;
       DISP_TIME : DURATION := 0.0;
       LINT_SEC  : INTEGER  := 0;

    begin
       accept START do
       null;
       end START;
       LINT_SEC := INTEGER(SECONDS(CLOCK));
       DISP_TIME := DURATION(LINT_SEC);
       while DISP_TIME < SECONDS(CLOCK) loop
          DISP_TIME := DISP_TIME + INTERVAL;
       end loop;
```

```
        loop
            delay DISP_TIME - SECONDS(CLOCK);
            for I in VELOCITIES'FIRST..VELOCITIES'LAST-1 loop
                VEL(I) := VEL(I+1);
            end loop;
            VEL(VELOCITIES'LAST) := VEL(VELOCITIES'LAST) +
                                    (0.012,0.0098,0.00275);
            exit when VEL(4)(X) > 700.0;
        end loop;
    end ACCELEROMETER;
end ATOD;


begin  -- main program

    ATOD.INITIALIZE_VELOCITY(INT_VEL);
    LINT_SEC := INTEGER(SECONDS(CLOCK));
    DISP_TIME := DURATION(LINT_SEC) + 0.8;
    ATOD.ACCELEROMETER.START;
    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
    P0(X) := POSITION(X);
    P0(Y) := POSITION(Y);
    P0(Z) := POSITION(Z);
    B(0)(X) := P0(X);
    B(0)(Y) := P0(Y);
    B(0)(Z) := P0(Z);

    POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
    DISP_TIME := DISP_TIME + INTERVAL;
    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
    P1(X) := POSITION(X);
    P1(Y) := POSITION(Y);
    P1(Z) := POSITION(Z);
    B(1)(X) := P1(X);
    B(1)(Y) := P1(Y);
    B(1)(Z) := P1(Z);

    POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
    DISP_TIME := DISP_TIME + INTERVAL;
    delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
    ATOD.GET_VELOCITIES(VELOCITY);
```

```
P2(X) := POSITION(X);
P2(Y) := POSITION(Y);
P2(Z) := POSITION(Z);
B(2)(X) := P2(X);
B(2)(Y) := P2(Y);
B(2)(Z) := P2(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P3(X) := POSITION(X);
P3(Y) := POSITION(Y);
P3(Z) := POSITION(Z);
B(3)(X) := P3(X);
B(3)(Y) := P3(Y);
B(3)(Z) := P3(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P4(X) := POSITION(X);
P4(Y) := POSITION(Y);
P4(Z) := POSITION(Z);
B(4)(X) := P4(X);
B(4)(Y) := P4(Y);
B(4)(Z) := P4(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
ATOD.GET_VELOCITIES(VELOCITY);
P5(X) := POSITION(X);
P5(Y) := POSITION(Y);
P5(Z) := POSITION(Z);
B(5)(X) := P5(X);
B(5)(Y) := P5(Y);
B(5)(Z) := P5(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
```

```
ATOD.GET_VELOCITIES(VELOCITY);
P6(X) := POSITION(X);
P6(Y) := POSITION(Y);
P6(Z) := POSITION(Z);
B(6)(X) := P6(X);
B(6)(Y) := P6(Y);
B(6)(Z) := P6(Z);

POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
DISP_TIME := DISP_TIME + INTERVAL;
while NO_TARGET = FALSE loop
   if POSITION(X) > 0.0 then

      ARRAY_IO.WRITE(C, B);

      delay (DISP_TIME - SECONDS(CLOCK) - 0.02);
      ATOD.GET_VELOCITIES(VELOCITY);
      P(X) := POSITION(X);
      P(Y) := POSITION(Y);
      P(Z) := POSITION(Z);

      POSITION := POSITION - SIMPSON(POSITION,VELOCITY);
      DISP_TIME := DISP_TIME + INTERVAL;

      P0(X) := P1(X);
      P0(Y) := P1(Y);
      P0(Z) := P1(Z);

      P1(X) := P2(X);
      P1(Y) := P2(Y);
      P1(Z) := P2(Z);

      P2(X) := P3(X);
      P2(Y) := P3(Y);
      P2(Z) := P3(Z);

      P3(X) := P4(X);
      P3(Y) := P4(Y);
      P3(Z) := P4(Z);

      P4(X) := P5(X);
      P4(Y) := P5(Y);
      P4(Z) := P5(Z);
```

```
            P5(X) := P6(X);
            P5(Y) := P6(Y);
            P5(Z) := P6(Z);

            P6(X) := P(X);
            P6(Y) := P(Y);
            P6(Z) := P(Z);

            B(0)(X)  := P0(X);
            B(1)(X)  := P1(X);
            B(2)(X)  := P2(X);
            B(3)(X)  := P3(X);
            B(4)(X)  := P4(X);
            B(5)(X)  := P5(X);
            B(6)(X)  := P6(X);
            B(0)(Y)  := P0(Y);
            B(1)(Y)  := P1(Y);
            B(2)(Y)  := P2(Y);
            B(3)(Y)  := P3(Y);
            B(4)(Y)  := P4(Y);
            B(5)(Y)  := P5(Y);
            B(6)(Y)  := P6(Y);
            B(0)(Z)  := P0(Z);
            B(1)(Z)  := P1(Z);
            B(2)(Z)  := P2(Z);
            B(3)(Z)  := P3(Z);
            B(4)(Z)  := P4(Z);
            B(5)(Z)  := P5(Z);
            B(6)(Z)  := P6(Z);
        else
            NO_TARGET := TRUE;
        end if;
    end loop;
    B(0)(X) := -1.00;
    ARRAY_IO.WRITE(C, ((B(0)(X),others => 0.0),(others => 0.0),(others => 0.0),
            (others => 0.0), (others => 0.0), (others => 0.0), (others => 0.0)));

end PROJ2;
```

```
# File: makefile

#   "make help" to print option list
#
#   Complete development cycle:
#     make family       -- makes Ada family and library directories
#     make              -- compiles, links, configures source
#     make run          -- run bootable code

MODE = s
PROC = 8
OPTS = /$(MODE) /t$(PROC)

# make the executable code
main.btl:   mainh.c$(PROC)$(MODE)   proj1h.c$(PROC)$(MODE)
proj2h.c$(PROC)$(MODE) main.pgm
    @ echo EXPECT 1 WARNING...
    iconf /s main.pgm
    @ f:\util\bell

mainh.c$(PROC)$(MODE):   proj0.o   proj0h.t$(PROC)$(MODE)
merger.t$(PROC)$(MODE) mainh.t$(PROC)$(MODE)
    ilink   mainh.t$(PROC)$(MODE)   proj0.o   proj0h.t$(PROC)$(MODE)
merger.t$(PROC)$(MODE) adarts8.lib hostio.lib occam8s.lib

proj0.o: common.ada proj0.ada
    ada invoke proj0.inv,yes

proj0h.t$(PROC)$(MODE): proj0h2.tax proj0h.occ
    occam $(OPTS) proj0h.occ

proj0h2.tax: proj0h2.occ
    occam /ta /x proj0h2.occ



merger.t$(PROC)$(MODE): merger.occ
    occam $(OPTS) merger.occ

mainh.t$(PROC)$(MODE): mainh.occ
    occam $(OPTS) mainh.occ
```

```
proj1h.c$(PROC)$(MODE): proj1.o proj1h.t$(PROC)$(MODE)
    ilink proj1h.t$(PROC)$(MODE) proj1.o adarts8.lib hostio.lib occam8s.lib

proj1.o: common.ada proj1.ada
    ada invoke proj1.inv,yes

proj1h.t$(PROC)$(MODE): proj1h2.tax proj1h.occ
    occam $(OPTS) proj1h.occ

proj1h2.tax: proj1h2.occ
    occam /ta /x proj1h2.occ

proj2h.c$(PROC)$(MODE): proj2.o proj2h.t$(PROC)$(MODE)
    ilink proj2h.t$(PROC)$(MODE) proj2.o adarts8.lib hostio.lib occam8s.lib

proj2.o: common.ada proj2.ada
    ada invoke proj2.inv,yes

proj2h.t$(PROC)$(MODE): proj2h2.tax proj2h.occ
    occam $(OPTS) proj2h.occ

proj2h2.tax: proj2h2.occ
    occam /ta /x proj2h2.occ

#
# misc.
#

run:
    iserver /sb main.btl

check:
    check /r

family:
    ada invoke family.inv,yes
```

```
-- File: main.pgm

#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"

#USE "mainh.c8s"
#USE "proj1h.c8s"
#USE "proj2h.c8s"

CHAN OF INT AdaChannel:
CHAN OF INT AdaChan:
CHAN OF SP FromFiler, ToFiler:

PLACED PAR

  PROCESSOR 0 T8

    PLACE FromFiler   AT link0.in:
    PLACE ToFiler     AT link0.out:
    PLACE AdaChannel  AT link2.in:

    [100000] INT ws1:
    main.harness (FromFiler, ToFiler, AdaChannel, ws1)

  PROCESSOR 1 T8

    PLACE AdaChannel  AT link1.out:
    PLACE AdaChan     AT link2.in:

    [100000] INT ws2:
    proj1.harness (AdaChannel, AdaChan, ws2)

  PROCESSOR 2 T8

    PLACE AdaChan AT link3.out:

    [100000] INT ws3:
    proj2.harness (AdaChan, ws3)
```

LIST OF REFERENCES

1. Swenson, E.N., Mahenske, E.B. and Stoutenburgh, J.S., "NTDS - A Page in Naval History", Naval Engineers Journal, pp.53-61, May 1988.

2. Uno R. Kodres, *Parallel Command and Decision Systems*, Technical Note, U.S.Naval Postgraduate School, February 1991.

3. G.M. Lundy, *Improving the Aegis combat system*, Technical Note, U.S.Naval Postgraduate School, July 1991.

4. INMOS limited, *The Transputer Family*, June 1986.

5. INMOS limited, *The Transputer Handbook*, October 1989.

6. INMOS limited, *Transputer Databook*, 1989.

7. INMOS limited, *Transputer Reference Manual*, October 1986.

8. INMOS limited, *Transputer Development and iq Systems Databook*, 1989.

9. Jack J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software*, Technical Report CS-89-85, University of Tennessee and Oak Ridge National Laboratory, September 1991.

10. Shem-Tov Levi and Ashok K. Agrawala, *Real Time System Design*, 1990.

11. Geoffrey C. Fox, Mark A. Johnson, Gregory A. Lyzenga, Steve W. Salmon, and David W. Walker, *Solving Problems on Concurrent Processors*, 1988.

12. Mike Tedd, Stefano Crespi-Reghizzi and Antonio Natali, *Ada for multi microprocessors*, Cambridge University Press, 1984.

13. Jan Skansholm, *Ada from the beginning*, Chalmer University of Technology, 1989.

14. Barnes, John Gilbert Presslie, *Programming in Ada*, Alsys ltd. 1989.

15. Booch, Grady, *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, 1986.

16. Thomas Richard McCalla, *Introduction to Numerical Methods*, 1967.

17. James A. Seveney and Guenter P. Steinberg, *Requirements Analysis for a Low Cost Combat Direction System*, Master Thesis, U.S.Naval Postgraduate School, Monterey, Ca., June 1990.

18. Alsys Limited, *Alsys Ada PC-Mothered transputer Cross-Compilation System*, User Manual, Newtown road, Henley-on-Thames, Oxfordshire, U.K., 1989.

19. Dennis D. Berkey, *Calculus second edition*, Boston University, 1988.

INITIAL DISTRIBUTION LIST

Defence Technical Information Center Cameron Station Alexandria, VA. 22304-6154	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA. 93943	2
Aegis Laboratory Room SP-531 Naval Postgraduate School Monterey, CA. 93943	2
Dr. Uno R. Kodres Code CS/KR Department of Computer Science Naval Postgraduate School Monterey, CA. 93943	5
Dr. Se-Hung Kwak Code CS/KW Department of Computer Science Naval Postgraduate School Monterey, CA. 93943	1
Commanding Officer Guided Missile Division, Ordnance Dept. Toong Prong, Sattahip Cholburi 20170 Thailand	1

Lt. Panurit Yuktadatta, RTN. 1
180/57 Chiwiwat 1, Soi Watbuahwan
Nonthaburi 11000
Thailand

Lt. Chatchai Chaowanasin 1
1504 Soi Watdan, Amphor Muang
Samutpraharn 10270
Thailand

Thesis
Y853 Yuktadatta
c.1 Simulation of a parallel processor based small tactical system.